DOCUMENT RESUME

ED 322 895 IR 014 594

AUTHOR	Yoder, Sharon Burrowes
TITLE	Introduction to Programming in Logo Using LogoWriter. Revised Edition.
INSTITUTION	International Society for Technology in Education, Eugene, OR.
REPORT NO	ISBN-0-924667-47-8
PUB DATE	90
NOTE	154p.
AVAILABLE FROM	International Society for Technology in Education, University of Oregon, 1787 Agate Street, Eugene, OR 97403-9905 ($14.95 prepaid).
PUB TYPE	Guides - Classroom Use - Guides (For Teachers) (052) -- Computer Programs (101) -- Guides - Classroom Use - Materials (For Learner) (051)
EDRS PRICE	MF01 Plus Postage. PC Not Available from EDRS.
DESCRIPTORS	*Computer Graphics; Elementary Secondary Education; Microcomputers; *Programing; Programing Languages; *Word Processing
IDENTIFIERS	ExperLOGO; *LOGO Programing Language; Turtle Graphics

ABSTRACT

This book is designed to teach LogoWriter, a programming language. Each of the 41 small chapters focuses on a particular LogoWriter word or idea, and provides examples for practice and experimentation. Topics included are: (1) getting started with LogoWriter; (2) using repeat and label; (3) using color and random, shapes and stamp, fill and shade; (3) using text, mixing text and graphics, writing procedures; (4) cut, copy, and paste; (5) working with pages, Logo grammar, Logo arithmetic, printing; and (6) text manipulation, interactive programming, and designing programs. Ten appendices include key summaries, copies of keyboard stickers, and a quick reference card. (DB)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

About the Author

Sharon Yoder has taught mathematics and computer science at the junior high and high school level for 15 years. Her most recent public school experience was as a secondary computer science teacher and a computer coordinator involved in developing system-wide computer curriculum and in planning teacher inservice training.

In addition, she has taught mathematics, computer science, and computer education at a number of universities in northeastern Ohio, including Kent State University, the University of Akron, and Cleveland State University. She has worked closely with the College of Education of Cleveland State University in developing their computer programming courses for teachers.

After a year as an Education Specialist for Logo Computer Systems, Inc., Sharon has returned to teaching. She is currently at the University of Oregon, where she teachers computer education courses.

For the past several years, she has conducted workshops and presented papers at conferences nation-wide, and has been involved in a number of book publishing projects, including the *Nudges* series. In addition, she has been a frequent contributor to *The Computing Teacher* and the *Logo Exchange*. She is currently editor of the *Logo Exchange* and a column editor for The Logo Center in *The Computing Teacher*.

Project Editor: Neal Strudler
Cover Design: Percy Franklin
Production: Susan Gorecki, Kerry Lutz, Tamara Kidd

The book prices given below are for prepaid orders. On other orders a $2.50 invoicing fee will be added. Please add shipping charges of $2.50 for orders up to $20; $3.50 for orders between $21 and $50; $4.50 for orders between $51 and $90; and 5% for orders above $90.

QUANTITY		**PRICE (U.S.)**
1-4	copies	$14.95
5-9	copies	$13.45
10+	copies	$11.95

Please place your orders with:

INTERNATIONAL SOCIETY FOR
TECHNOLOGY IN EDUCATION
University of Oregon
1787 Agate Street
Eugene, Oregon 97403-9905
(503)346-4414, 346-4429
CompuServe: 70014,2117 BITNET: ISTE@Oregon

Revised Edition
Copyright © ISTE 1988, 1990
ISBN 0-924667-47-8

Introduction to

Programming in Logo

Using LogoWriter

Revised Edition

by

Sharon Yoder

Table of Contents

Introduction 1
Overview of the Booklet 1
1. Getting Started with LogoWriter 3
2. Using Repeat and Label 7
3. Color and Random 11
4. Shapes and Stamp 15
5. Defining New Shapes 17
6. Fill and Shade 19
7. Using Text 21
8. Mixing Text and Graphics 25
9. Writing Procedures 27
10. More Than One Procedure 31
11. A Word About Designing Programs 35
12. Using PRINT 37
13. Printing Text on a Printer 39
14. Cut, Copy, and Paste 41
15. Music 47
16. Multiple Turtles 51
17. Logo Grammar 53
18. Using READCHAR 57
19. Indefinite Repetition 61
20. Animation 63
21. Creating Definite Loops 67
22. Printing Text on the Screen 69
23. More Interactive Programming 71
24. Working with Pages 75
25. More on Making Decisions 77
26. Procedure Inputs 81
27. More on Multiple Turtles 85
28. Using Inputs to Pass Values Among Procedures 89
29. Incrementing 93
30. Logo Arithmetic and Math 101
31. OUTPUT 103
32. Moving the Cursor from Procedures 107
33. Using Editing Commands from Procedures 109
34. More on Text Manipulation 111
35. Using MAKE 115
36. Cartesian Commands 119
37. Predicates 123
38. FIRST, BUTFIRST, LAST, and BUTLAST 127
39. ITEM and COUNT 131
40. WORD, SENTENCE, and LIST 135
41. FPUT and LPUT 137

Appendix 1: Setting Up 139
Appendix 2: STARTUP 141
Appendix 3: Chaining Pages 141
Appendix 4: Using Tools 143
Appendix 5: WHEN 143
Appendix 6: ProDOS and IBM Dos Primitives 145
Appendix 7: Exchanging Files 147
Appendix 8: LogoWriter Keys 149
Appendix 9: Shapes Pages 157
Appendix 10: Quick Reference 159

Introduction

The Logo programming language was originally developed at MIT in 1968 by Seymour Papert and his colleagues. It was initially implemented on mainframe, timeshared computer systems. Those versions did not include the familiar turtle graphics that are now considered an integral part of Logo. In the early 1980's, Logo was implemented on microcomputers and began to appear in classrooms across the country. Since those early implementations, there have been many versions of Logo for many different computers. In this booklet, you will be learning about the version of Logo called LogoWriter. This version differs from earlier versions in that a word processor is an integral part of the system and the user interface has been redesigned so that it better matches the perceptions of novice users. That is, the human-machine interface has been modified so that many things that users will "naturally" do and/or assume are correct will indeed be correct. However, most of what you will learn in this booklet about the Logo programming language applies to any version of Logo.

Overview of this Booklet for Teachers

The goal of this booklet is to teach about programming in the Logo language. However, you need to be aware that there is much more to Logo than just learning to program using it. Logo encompasses a philosophy of learning, best explained in Seymour Papert's book , *Mindstorms: Children, Computers, and Powerful Ideas*. You should balance your learning and/or teaching of Logo programming techniques with learning about and/or applying the Logo philosophy using sources such as the writings of Papert.

Problem solving has long been a component of the teaching Logo. This book does not focus on specific problems solving ideas except as they affect the program writing and debugging process. It seems appropriate that you include some specific teaching of problem solving as you work with Logo. You can find ideas for such lessons in the book *Getting Smarter at Solving Problems* by Dave Moursund. This book is available through ISTE at the address given in this book.

Introduction to Programming in Logo Using LogoWriter is a textbook-like source of information about Logo syntax and grammar. Although it is not as technical as a reference manual, it does contain detailed explanations of most Logo primitives. However, it goes further in that it discusses some computer science concepts such as programming style, modularity, and control structures. As a teacher of Logo or trainer of Logo teachers, you need to be comfortable enough with Logo to help your students debug their programs. This book is designed to help you learn Logo syntax in a rather structured manner.

If you are a teacher of younger students, you should keep in mind, however, that this booklet is in *no way designed to model the way you should teach Logo in the classroom.* Your teaching should encompass the exploratory, discovery learning emphasized in the Logo philosophy. Most likely you will not want to teach a formal Logo lesson each day. On the other hand, if you are teaching older students, say in a computer science class, you may want to use a more structured approach, or perhaps combine the two approaches. Whatever way you teach Logo, you should be aware that there is nothing magical about the sequence in which topics are presented in this book. There are many appropriate paths available for learning about Logo.

If you are interested in pursuing your study of Logo as a programming language, then you should explore the book *Computer Science Logo Style, Volume 1* by Brian Harvey. This book, the first of a series of three books on Logo and computer science, is available from MIT Press in Cambridge Massachusetts.

Overview of the Book for Learners

This book is designed to teach you LogoWriter in a rather structured manner. The major portion of the the book is divided into small chapters, each of which focuses on a particular LogoWriter word or idea. You should not simply read the chapters in the book, but rather should try out examples that are given in the text and experiment with them until you understand the concept being introduced. Most chapters end with some hints to make your work with Logo easier and some ideas for "projects." These project ideas are designed to give you an opportunity to experiment with the ideas you have learned and to extend your knowledge of LogoWriter. You should take some time to work through some of the ideas in each chapter before moving to the next section of the book.

Note that the book contains a fair amount of reference material. In the appendices you will find key summaries, copies of the keyboard stickers that come with the LogoWriter kits, a copy of the "shapes" pages, and a "quick reference card" that includes a list of all of the LogoWriter primitives. (The word "primitive" refers to a word that LogoWriter "knows.") You may want to mark your book so that you can easily locate these references, or perhaps make copies of them for easy reference. It is not appropriate to memorize all of the keys needed to work with LogoWriter or all of the Logo primitives. Rather you should have on hand a reference such as these pages in which to look up the information that you need.

1. Getting Started with LogoWriter

STARTING LOGOWRITER

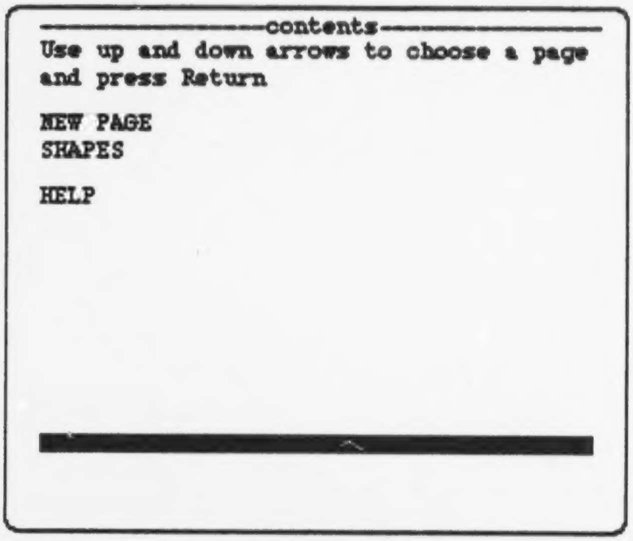
To begin you will need

- A LogoWriter Master Disk (containing the LogoWriter language).
- A LogoWriter Scrapbook disk (this is a "files" disk).

Put the Logowriter Master Disk in the disk drive of the computer and then start the system. (The specific keys to press depend on the kind of computer that you are using. Check with your instructor or someone knowledgeable about the hardware you are using if you have questions.) Also see Appendix 1.

The disk drive will whir and buzz and then you will see a screen containing the word LogoWriter written in large letters.

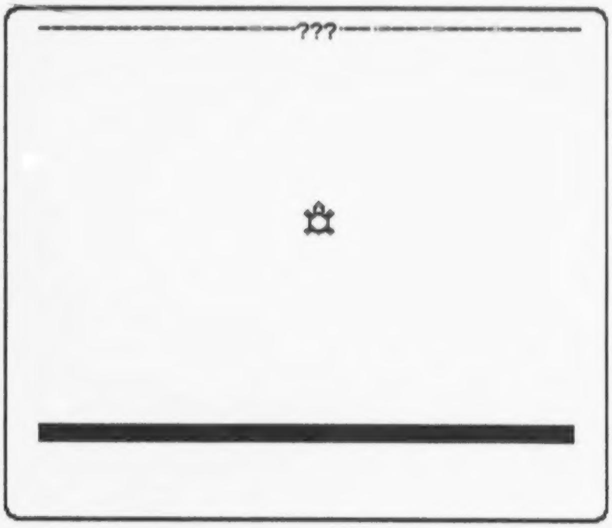
Take out the Logowriter Master Disk and put it away. Put in your Scrapbook Disk, and press the Return/Enter key. You then see the contents page which shows a list of pages that are on your disk.



```
----------------contents----------------
Use up and down arrows to choose a page
and press Return

NEW PAGE
SHAPES

HELP
```

To select a page from the Scrapbook disk, you use the arrow keys to move the small square, called the cursor. When the cursor is on the page name you wish to use, press Return/Enter. In this case, select NEW PAGE and press Return/Enter.

Now the fun begins! Your screen should look like this:



This is a blank LogoWriter page for you to work with. Whenever you get a new page, it is a good idea to give it a name. To do this, you must type

`NAMEPAGE "name.you.want` (Return/Enter)

or

`NP "name.you.want` (Return/Enter)

If you make a mistake in typing, Logo will respond with a message such as

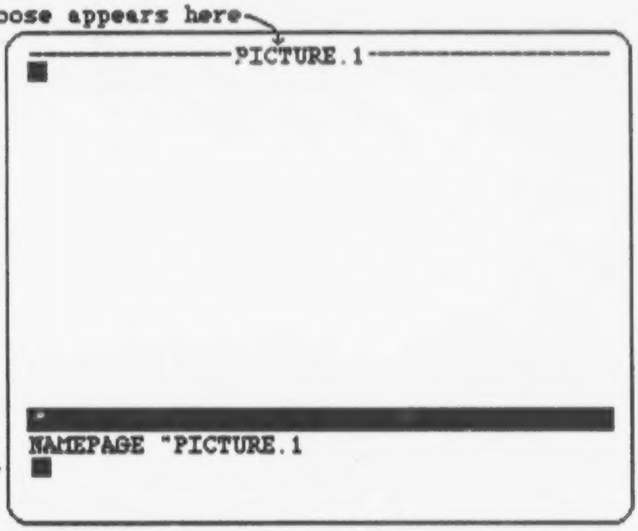
```
I don't know how to ...
```

Simply try again, paying particular attention to the spaces.

Note that name.you.want is the name you want to use for the page. LogoWriter will tell you if you try to use an unacceptable name. Be sure to put a " (quote mark) before the name. Do not leave a space after the name.

For example, you might type

`NP "PICTURE.1` (Return/Enter)



The name you choose appears here

```
PICTURE.1
```

```
NAMEPAGE "PICTURE.1
```

This cursor is flashing

Notice that you type the name with a single quotation mark (") in front of it and that there are no spaces in the name. Instead, a period is used in place of spaces to make the name easier to read. You can name your page almost anything you wish. LogoWriter displays a message telling you if you use illegal characters or if your name is too long. The name you type appears at the top of the page in place of the three ???'s.

DRAWING WITH THE TURTLE

Now you are ready to start drawing with the turtle. The turtle holds a pen which draws lines as the turtle moves. Try typing

`FORWARD 50` (Return/Enter)

or

`FD 50` (Return/Enter)

Be sure you type a zero (0), not an O ("oh"). Next type

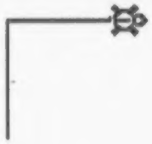
`RIGHT 90` (Return/Enter)

or

`RT 90` (Return/Enter)

Notice how the turtle turns. Using the up arrow key, move the cursor that is at the bottom of the screen until it is on the "FORWARD" line. Press Return/Enter.

Does your turtle drawing look like this?



Do you see that you don't have to retype a command to use it again? Can you finish drawing the square without typing any more commands?

Next type

`CG` (Return/Enter)

to clear the screen. CG stands for Clear Graphics.

What happens if you type

`FORWARD`

and press Return/Enter? Logo is telling you that you must put a number after FORWARD to tell the turtle how far forward to move. Logo usually gives you helpful messages when you make a mistake. Read them carefully.

Below is a list of some of the commands that the turtle understands:

```
FORWARD number          BACK number
FD number               BK number

RIGHT number            LEFT number
RT number               LT number

PU                      PD
(Pen Up)                (Pen Down)

CG
(Clear Graphics)
```

Experiment with these commands to see how each one works. Remember to press Return/Enter after each command. After you have experimented for awhile, decide on a design or drawing to make. If you make mistakes you can use CG to clear the page and start again. Recall that the commands you used are still available at the bottom of your screen. Remember that this area of the screen is called the Command Center.

Whenever you have something that you want to keep, press Esc. Esc causes LogoWriter to make your page a part of the Scrapbook. That is, it puts it on your Scrapbook disk. Recall that the page in LogoWriter is above the solid line; the commands you typed below that solid line are in the Command Center. The Command Center is not saved with the page. You will learn how to keep save commands that you have typed later in this book (Chapter 9)

Once you have saved your page by pressing Esc, simply select your page from the Contents page and you will see that it looks exactly as it did when you left it.

TIPS AND TECHNIQUES

Always name your page before you begin work. If you don't like the name, you can easily change it by using the NAMEPAGE command again.

When using NAMEPAGE, you must always have a space and then a single quotation mark (") before the name of the page. There is no space after the quotation mark. There is no second quotation mark after the name.

Spaces are very important in LogoWriter. If you get error messages that you don't understand, look carefully at what you typed. For example, you may not have any spaces in a page name. On the other hand, you must have spaces between a command like FORWARD and the number after it.

Beginners sometimes find the differences between the commands FORWARD and BACK and the commands RIGHT and LEFT difficult to understand. FORWARD and BACK are movement commands. They cause the turtle to take "turtle steps." RIGHT and LEFT are turn commands. The turtle doesn't move at all. Instead it turns about its middle, like a ballet dancer spinning on her toe.

It is not unusual to want to erase lines that you draw by accident. It is possible to do this using PE (Pen Erase). You type PE and draw over then line you don't want. Then you must type PD to continue to draw. Unless you erase a line immediately after drawing it, it can be very difficult to erase it completely. Generally it is better to start over rather than to try to fix several small errors on a drawing.

When you are finished working with LogoWriter, always be sure that you see the Contents Page on your screen before you turn off the computer or let someone else use it. You must press Esc to get to the Contents Page. This means that any work you have done is saved. (You can always erase unwanted work at a later time.)

Important note: When you save a page using Esc, only the graphics and letters above the solid line are saved. Text in the Command Center is not saved when you press Esc.

PROJECT SUGGESTION

If you can't think of any other kind of design to draw, you might want to try having the turtle draw your initials. When you have a design that you are satisfied with,

- Be sure it is saved (press Esc).
- Print it on paper by typing PRINTSCREEN.

NOTE: Your disk must be configured for use with the printer you are using. See Appendix 1.

2. Using REPEAT and LABEL

You have learned how to make pictures with the turtle. You have also learned how to save pages on your Scrapbook. Now you can begin adding more Logo commands to your new vocabulary.

USING REPEAT

Get a new page, name it, and then type

```
FORWARD 50
RIGHT 90
```

Remember to press Return/Enter after each line. Also remember that you can use the up arrow and the Return/Enter key to repeat these movements over and over. However, there is an easier way to have LogoWriter repeat actions. Type

```
REPEAT 4 [FORWARD 50 RIGHT 90]
```

What happens?

REPEAT is a new command. It must be followed by a number and a list of commands. Notice that the list of commands is contained in square brackets ([and]). Try something different. Type

```
CG
REPEAT 90 [FORWARD 80 RIGHT 178]
```

Don't forget to press Return/Enter.

You get a "fluffy" ball. How about

```
CG
REPEAT 18 [FORWARD 20 RIGHT 45
                  BACK 10 LEFT 25]
```

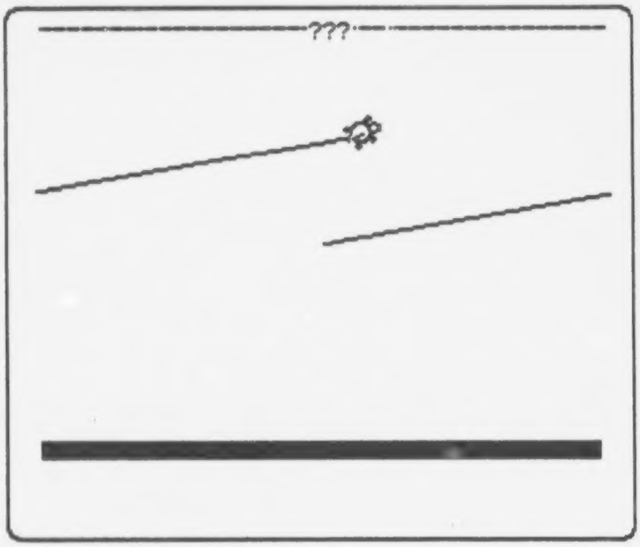
or

```
CG
REPEAT 360 [FORWARD 1 RIGHT 1]
```

Have you discovered what happens when the turtle goes off the edge of the page? If you type

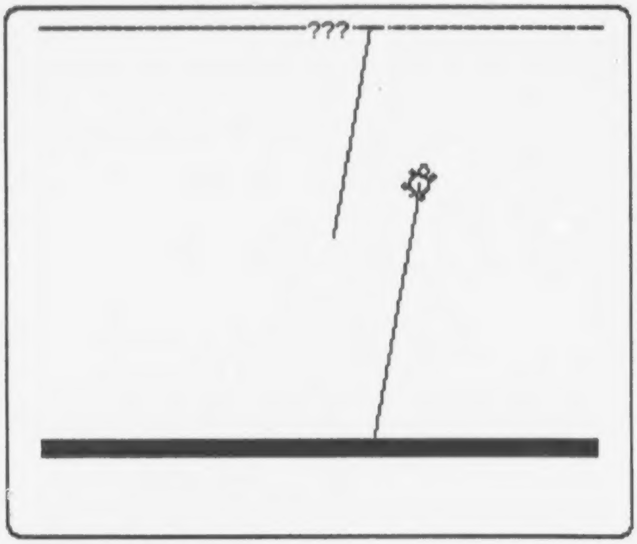
```
CG
RIGHT 80
FORWARD 300
```

you see that when the turtle goes off the page on the right, it reappears again on the left at the same distance from the bottom of the page. This is called "wrapping".



If you type

```
CG
RIGHT 10
FORWARD 200
```



you see that it wraps around the top and bottom of the page as well.

If a design you make would look better without the turtle showing, you can type

```
HT
```

for Hide Turtle. When you want to see the turtle again, you can type

```
ST
```

for Show Turtle.

Take some time to experiment with REPEAT. You can get some fascinating patterns and designs. Don't forget to press Esc if you create a page you want to keep.

RENAMING A PAGE

What if you save a design on your page by pressing Esc and then you change what is on the page and want to keep that *new* design as well? If you press Esc, the newest design will erase the previous one. *Esc always saves the current page using the name at the top of the page.* You can solve this problem by renaming the page.

For example, suppose you create page PICTURE.1 and press Esc. Then, if you select PICTURE.1 from the Contents page again and change the design on the page to something you want to keep, you must change the name of the page in order to save the new design. Type

```
NAMEPAGE "PICTURE.2
```

You will see the new name at the top of the screen change.

Next, press Esc to save the new page. When the Contents page appears, you see the names of both of the pages. You can then choose whichever you want, or a new page.

LABELING A DRAWING

Be sure that you have saved everything that you want to keep. Then type

```
CG
```

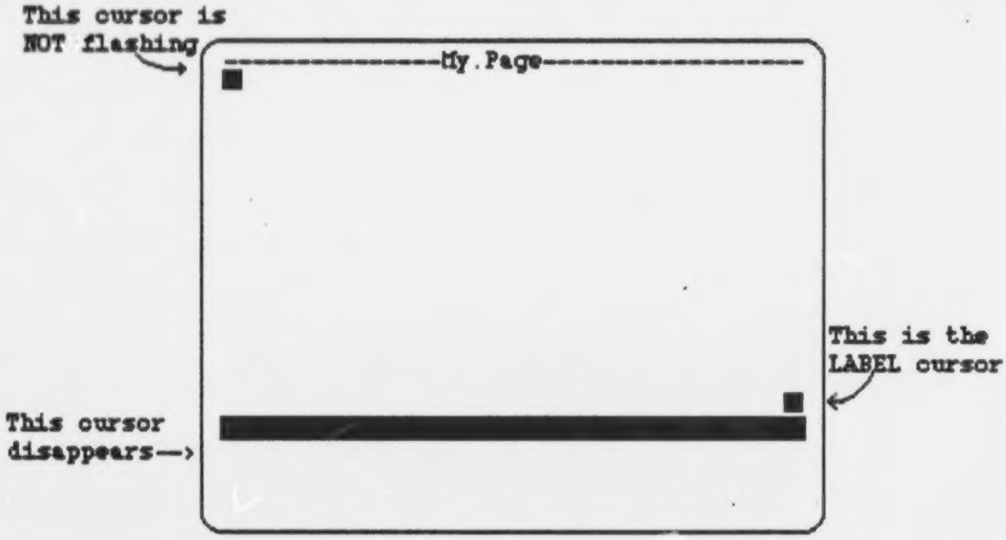
Next, hold down the Label keys. (Check the key reference sheet in Appendix 8 for the keys to use for your computer.)

Notice that a cursor appears in the lower right corner of the screen. (See the diagram at the bottom of the page.) Use the arrow keys to move the cursor around on the page. Try typing a word. It appears next to the cursor. You are in Label mode. With Label mode you can attach graphics text to your drawings. *When you are done typing words, press Esc to "wake up" the cursor in the Command Center.*

Perhaps, as you experimented with Label mode, you discovered that you couldn't erase mistakes in the way you expected. To learn how to fix errors in Label mode, first type

```
CG
```

press the Label keys, and then move the cursor to the middle of the page. Now type



```
CTAS
```

Oops. That was supposed to be CATS. Using the arrow keys, move back to the T. Then type T. The T disappears. Next type A. The A disappears. Now you can correct the word by typing AT in the correct spot. Experiment with correcting errors in Label mode. When you are done, press Esc to get back to the Command Center. If you want to get rid of the letters, type

```
CG
```

to get a clear page. Keep in mind that Label produces text that is part of the graphics since Clear Graphics removes it. Later you'll learn about text that is not part of the graphics.

TURTLE MOVE

You have been moving the turtle using the FORWARD, BACK, LEFT, and RIGHT commands. There is another way to move the turtle on the page. Hold down the Turtle Move keys. (See Appendix 8 for the keys to use with your computer.) Now use the arrow keys to move the turtle. See the diagram at the bottom of the page. *To leave Turtle Move, press Esc.* When the cursor is again flashing in the Command Center, type

```
REPEAT 4 [FORWARD 50 RIGHT 90]
```

You now have a square that is not in the center of the page. You can use Turtle Move to place designs wherever you want on the page.

TIPS AND TECHNIQUES

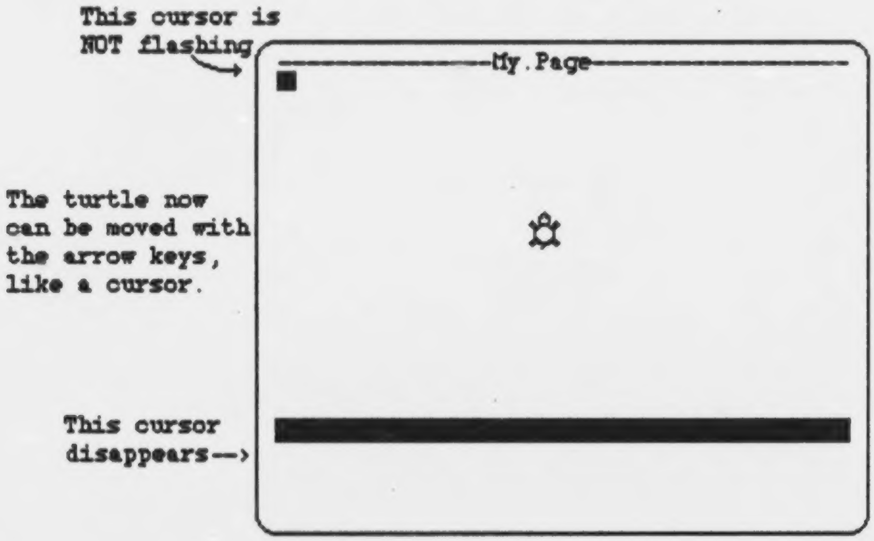
You must use *square brackets*: [and] with REPEAT, not parentheses: (and).

When using Label or Turtle Move, it is easy to "lose" your cursor. As you work with these new ideas, pay close attention to the cursor. There is a cursor in the upper left hand corner of the page which is not flashing. (You will learn about that cursor later.)

If the cursor in the Command Center at the bottom of the screen disappears, then you are in Turtle Move or Label Mode. Press an arrow key. If the turtle moves, then you are in Turtle Move. If there is a second cursor on the page (above the solid line) and it moves when you press the arrow keys, then you are in Label mode. To get back the cursor in the Command Center, press Esc.

Don't worry if you press Esc too often. If you have named your page, LogoWriter will simply save your work and put you on the Contents Page. Select your page and continue your work. If you have not named your page, LogoWriter will ask you to name it. If you wish, you can ignore this message and continue your work.

Keep in mind that Label should be used only for short pieces of text. You should not use it to write stories or even long sentences. In Chapter 7 you will learn how to use the word processor to write a lot of text.


This cursor is NOT flashing
My.Page
The turtle now can be moved with the arrow keys, like a cursor.
This cursor disappears

PROJECT SUGGESTIONS

Now that you know about Label as well as Turtle Move, create a drawing that is not in the center of the page and label it. When you are satisfied with your drawing, be sure to press Esc to save it and then type PRINTSCREEN to print it on the printer.

If you can't think of anything else, use REPEAT to make a flower in the upper left corner of the page and then use Label to put a few words describing the flower in the lower right corner of the page.

Color and Random

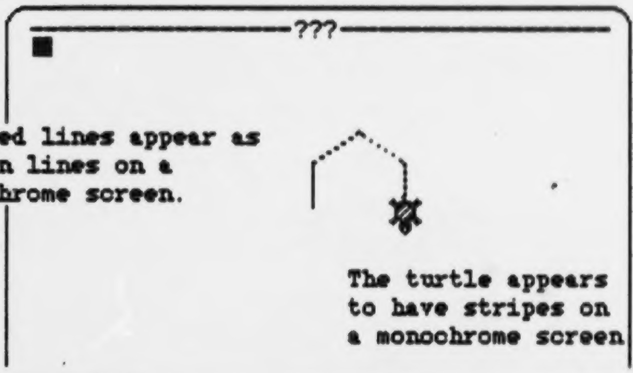
DRAWING IN COLOR

So far you have created drawings with the turtle that were white on a black background ... unless you were using a monochrome screen! The turtle can also draw in colors. Try this series of commands and watch carefully what happens. (You might also want to jot down the numbers you use and what colors they represent.)

```
SETC 2
FORWARD 30
RIGHT 60
SETC 3 <— Use the arrow keys to move up and then
FORWARD 30  the Delete key to change just the
RIGHT 60    number.  Then you don't have to
SETC 4      retype the FORWARD and RIGHT
FORWARD 30  commands.
RIGHT 60
SETC 5
FORWARD 30
RIGHT 60
SETC 1
FORWARD 30
RIGHT 60
SETC 0
FORWARD 30
RIGHT 60
```

Did you notice that the turtle changes to the color of pen it is carrying? What is the color number for white? What is the color number for black? What happens if the turtle turns black? What other color numbers did you discover? How many colors are available on the computer you are using? Keep in mind that the color of the turtle represents the color of the *pen* it is carrying and thus the color of the line the turtle will draw.

NOTE: If you are using a monochrome screen and change the color of the turtle, the turtle will appear to have stripes and the lines that are drawn will be more broken.



Colored lines appear as broken lines on a monochrome screen.

The turtle appears to have stripes on a monochrome screen

CHANGING THE BACKGROUND COLOR

Not only can you change the color of the pen using SETC, but you can also change the color of the background. For example,

```
SETBG 3
```

changes the color of the entire page. Experiment with different background colors.

On some kinds of computers, you will discover that when you draw on a colored background with a colored pen, you occasionally get some odd effects. The colors aren't what you expect them to be. That is not something you are doing wrong. It has to do with how color is represented inside your computer. For best results you will want to use black or white pens on colored backgrounds and use colored pens only on black or white backgrounds.

USING RANDOM

Everything that you have told the turtle to do so far has been exact: "go forward 50 steps," "turn left 57 degrees," or "draw in pen color 3." It is also possible to have the turtle behave in ways that you cannot totally predict. You can use LogoWriter to create random numbers. Random numbers are numbers that have no apparent pattern to them. The primitive RANDOM is used to create these numbers.

RANDOM is different from the Logo primitives that you have seen so far. If you type

```
RANDOM 50
```

LogoWriter responds with

```
I DON'T KNOW WHAT TO DO WITH
                     some.number
```

This response occurs because RANDOM is a reporter. It returns a value; reports it back to Logo. Repeat the statement

```
RANDOM 50
```

a number of times. It is likely that each time you do so, you will get a different number after "I DON'T KNOW WHAT TO DO

To make use of the RANDOM reporter, you must tell Logo what to do with the number it produces. Try, for example

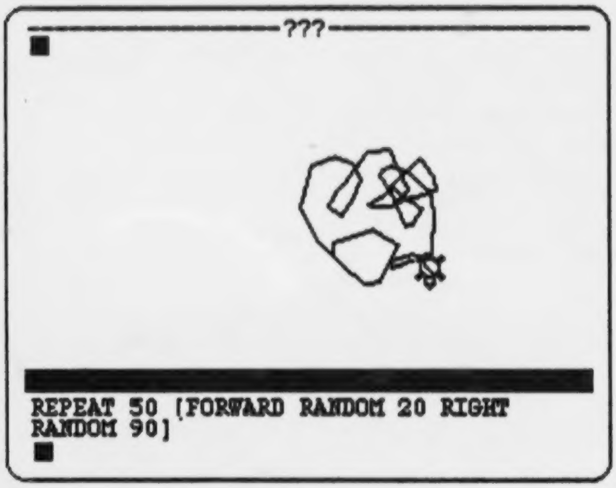
```
FORWARD RANDOM 50
```

Repeat this statement a number of times. Do you see that the turtle moves different amounts at different times? RANDOM reports to FORWARD some number from 0 to 49 and then the turtle moves forward that number of "turtle steps." That is, if RANDOM 50 reports the number 37, then the turtle follows the command FORWARD 37. If RANDOM 50 reports 11, then the turtle moves forward 11 "turtle steps."

You can send the turtle on a random walk by typing something like

```
REPEAT 50 [FORWARD RANDOM 20 RIGHT
          RANDOM 90]
```

Did your design look anything like this?



```
REPEAT 50 [FORWARD RANDOM 20 RIGHT
RANDOM 90]
```

Experiment with RANDOM. Can you get some other interesting effects using RANDOM?

How about picking random colors? That certainly is possible. Recall that the colors are numbered beginning with zero. So, if you use RANDOM 6, it will report a number between 0 and 5 – exactly what you want. You can get flashy effects by typing:

```
REPEAT 10 [SETBG RANDOM 6 WAIT 5]
```

Or, you can change the background color each time the turtle draws a line. Try this:

```
CG
SETC 1
REPEAT 4 [SETBG RANDOM 6 FORWARD 50
         RIGHT 90]
```

Try experimenting with random pen colors. You can get some beautiful patterns by simply changing the pen color randomly each time you move the turtle. For example,

```
CG
REPEAT 4 [SETBG RANDOM 6 SETC RANDOM 6
         FORWARD 50 RIGHT 90]
```

When you are working with color, the easiest way to reset the screen is to use

```
RG
```

Reset Graphics. This command puts the turtle in the center of the page, sets the background to black, and the pen color to white.

If you are using the GS version of LogoWriter, there are a number of new primitives for working with color. CHANGECOLOR allows you to modify the color associated with a given color number. CHANGECOLOR takes to inputs: the color number you want to change and a list of three numbers (each 0 - 15) that selects the intensity of red, green and blue. Thus typing

```
CHANGECOLOR 2 [15 15 0]
```

changes color number 2 to have intensity 15 for its red and green components and 0 for it's blue component. To find the color values associated with a given number, you can type

SHOW COLORVALUE *number*

COLORVALUE returns the list of intensities associated with *number.* To get all of the colors back to normal, you can type

```
RESETCOLORS
```

TIPS AND TECHNIQUES

If you try several different numbers after SETBG and you see no color, then you are using a monochrome screen. There is no way to get color on a monochrome monitor.

Using RANDOM is a bit like drawing numbers "out of a hat." You cannot predict what numbers will turn up next. If this is a new idea for you, experiment with the commands

```
CG
FORWARD RANDOM 50
```

until you see that the turtle moves a different amount each time.

Keep in mind that you can save yourself a lot of typing time by using the arrow keys to scroll up and down in the Command Center. You can also use the left and right arrow keys to move back and forth in a line that you want to change.

PROJECT SUGGESTIONS

Create a page using color and RANDOM. You might

- Send the turtle on a random walk with a random pen color at each move.
- Draw geometric figures with flashing random background colors.
- Fill the screen with randomly colored stars.

When you are finished, be sure to press Esc to save your page. You may also want to print a copy using PRINTSCREEN.

4. Shapes and STAMP

CHANGING THE SHAPE OF THE TURTLE

Have you noticed the page SHAPES on the Contents Page? Maybe you even looked at it! If not, select SHAPES and you will see a screen filled with small pictures. Numbers 1 - 10 are just dots. Numbers 11 - 20 are a variety of shapes. (On the "primary" Scrapbook disk, numbers 1 - 25 are a variety of shapes; 26 - 30 are blank.) Is there one that you particularly like? Remember its number. (See Appendix 9 for pictures of all of the shapes.) Press Esc to leave the shapes page and then get a new page.

Once you see the turtle on the screen, type

```
SETSH number.you.picked
```

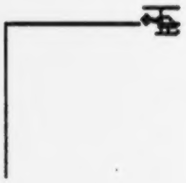
Surprise! The turtle is no longer a turtle! Try another number. And another. If you want the turtle shape back, type

```
SETSH 0
```

The turtle can become any of the shapes shown on the SHAPES page.

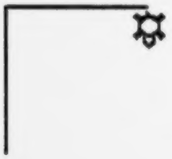
Next, with the turtle wearing a "costume" other than its familiar turtle shape, type

```
CG
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
```



Do you notice anything unexpected? If not, type

```
CG
SETSH 0
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
```



and compare the movement of the turtle shape and the other shapes. Do you see that the turtle shape turns in the direction that it is moving, but the other shapes do not? That can be a problem when you are trying to draw with the turtle and forget which way it is facing. Of course, you can always use

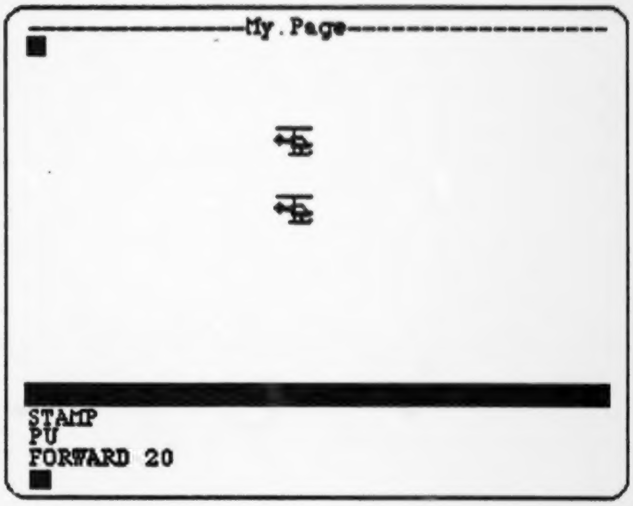
```
SETSH 0
```

to see where the turtle is facing. In fact it is a good idea to leave the turtle in its normal "turtle costume" until you get a shape placed where you want it on the page.

STAMPING THE TURTLE SHAPE

Not only can the turtle draw, but it can behave like the rubber stamps that people use to stamp dates or mark packages. The turtle can stamp only its current shape. Try this.

```
CG
SETSH number.of.your.choice
PD
STAMP
PU
FORWARD 20
```



```
STAMP
PU
FORWARD 20
```

You should now have two "copies" of the turtle shape. Which one is "real"? Type

```
FORWARD 20
```

and you will see that one of the turtles moves and the other stays put. Another technique for discovering which turtle is active and which is a stamped copy is to use HT. The active turtle will disappear and the stamp will not change. You can then see the active turtle again by typing ST. As you can see, the stamp of the turtle shape will remain on the screen until you type CG.

Take note of the sequence of commands to get a copy of the turtle:

- Put the pen down.
- STAMP the shape.
- Lift the pen (so there is no trail).
- Move the turtle (so you can see the stamped copy).

You can stamp any number of shapes in any number of places on the screen. Try it!

TIPS AND TECHNIQUES

When you are working with different turtle shapes, you can't tell which way the turtle is facing. When you want to draw with the turtle, type

```
SETSH 0
```

to use the turtle shape. When you have finished drawing, then change to the turtle shape you want to appear on the page.

If STAMP does not seem to work, you probably forget to put the pen down. Get into the habit of using the sequence

```
PD
STAMP
PU
```

so that STAMP works correctly and you don't accidentally draw lines after you stamp a shape.

In some versions of LogoWriter, if you stamp a turtle on top of an already stamped turtle shape the turtle will disappear. If you move the turtle on top of a stamped turtle shape, both may disappear. Don't panic. Just move the turtle. (Note that the behavior of the turtle here is much like Label mode.)

You can erase a stamped turtle shape by setting the pen color to match the background color and stamping on top of the shape you don't want. It can be difficult to get a shape in exactly the right place, however. Sometimes it is easier to use the square shape (number 11 on the Intermediate version of LogoWriter) as an eraser. You might type

```
SETSH 11
```

then use Turtle Move to place the turtle, set the pen color to the same color as the background, and then stamp the shape.

```
SETC BG
PD
STAMP
PU
```

These steps can be confusing because you can't see the turtle after you type SETC BG. When you have finished typing PU, be sure to set the pen color (SETC) to some other color, like white. Incidentally, this same technique can be used to remove lines you no longer want in your drawing.

PROJECT SUGGESTIONS

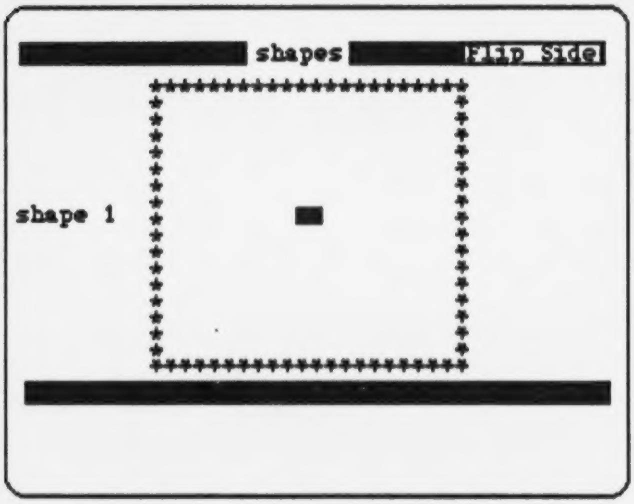
By combining designs made with the turtle, stamped shapes and Label mode, you can create a variety of interesting pages. For example, you might create a

- Postcard with a picture and a greeting.
- Valentine's card bordered with hearts.
- Sign to post which advertises the next dance.
- Cover for a report.
- Campaign poster.
- Sign to go on the locker of a friend.
- A reminder about an assignment schedule.
- An overhead to be used for class or for a report.

5. Defining New Shapes

When you were working with the Shapes page in the last section, did you wonder why part of the shapes were just dots? In this section you will get the answer.

Go to the Shapes page, and use the Flip keys, to flip to the back of the Shapes page. (See Appendix 8 for the keys used on your computer.) The "back" of the page shows you an enlargement of each shape.



Use the Next Screen and Previous Screen keys to move from shape to shape (See Appendix 8). Take a look at all of the shapes.

Go back to shape number 1. Use the arrow keys (Up, Down, Right, and Left) to move around within the shape. How many "blocks" across is the shape? How many up and down?

Next, press the space bar. If you are on a "colored in" block, it disappears. If you are not on a block, a block appears. See if you can make one of your initials in the shape grid.

When you have finished your initial, flip the page back. Do you see that your initial appears on the front of the Shapes page now? Press Esc to leave the Shapes page, get a new page, and type

```
SETSH 1
```

The turtle is now the shape that you created! Your new shape has become part of the Shapes page. Name your page, then type

```
SHAPES
```

to go back to the Shapes page. Pick one of the predefined shapes and change it. Perhaps you'd like to change the way the kitty looks or make the car a bit more sporty. When you are finished, flip the page to see your revised shape. When you press Esc, LogoWriter returns you to the page you just named. Try using your revised shape.

RESTORING A SHAPES PAGE

You can change any or all of the shapes on the Shapes page to whatever you want. However, you must remember that when you replace a shape with the new one, the old one is lost. Think carefully before you press Esc after changing shapes.

If you accidentally destroy the shapes that came with your version of LogoWriter, you can restore them using another LogoWriter disk with the correct Shapes page. Be *very* careful when you copy an entire Shapes page; it is easy to accidentally destroy the page you want to keep.

First, be sure you are on the Contents page.

1. Put the disk with the correct Shapes page in the disk drive and select the Shapes page.
2. Remove the disk with the correct Shapes page.
3. Put the disk onto which you want to copy the correct Shapes page.
4. Press Esc.

Both disks now contain identical Shapes page. Later in this book (Chapter 14) you will learn how to copy individual shapes from one place to another.

PROJECT SUGGESTIONS

Now that you can create any shapes you want, try one of the projects from the previous section, but with your own shapes. For example, you can make

- More personalized greeting cards.
- Signs with special symbols on them.
- Cartoons

You can then print your results with PRINTSCREEN.

6. FILL and SHADE

You now know how to change the pen color and the background color. With LogoWriter you can also fill in areas on the page with the current pen color. You can also fill areas of the page with turtle shapes – even with shapes that you have created!

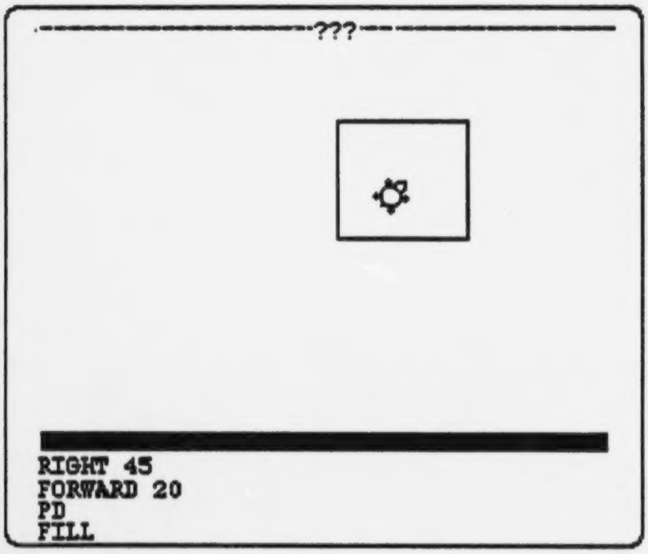
Clear your page and type

```
REPEAT 4 [FORWARD 50 RIGHT 90]
```

to draw a square on the screen. Next type

```
PU
RIGHT 45
FORWARD 20
PD
```

the turtle is now inside the square.



```
RIGHT 45
FORWARD 20
PD
FILL
```

Now type

```
FILL
```

The square fills with color. Type

```
CG
SETC 5
```

Use the arrow keys to move up to the REPEAT line. Press Return/Enter to run each line below the REPEAT. Now the turtle draws a colored square and fills it with color.

Notice that the pen was lifted before moving the turtle into the square. If you leave the pen down, move the turtle and then type FILL, nothing will happen. This is because the turtle "senses" that it is already on top of a colored spot. Always be sure to lift the pen before moving the turtle into an area to be filled.

You can fill any closed area with whatever color you want. Be careful, however, that the area really is closed. Otherwise the color may "leak out" all over the page! Occasionally FILL does not work as you expect. If you have difficulty with FILL, move the turtle just a few turtle-steps and then try again. Remember, keep the pen up when moving the turtle; put the pen down when you are ready to fill.

Now type

```
CG
REPEAT 4 [FORWARD 80 RIGHT 90]
PU
RIGHT 45
FORWARD 20
PD
```

Again you see a square with the turtle inside. Next type

```
SETSH 29
```

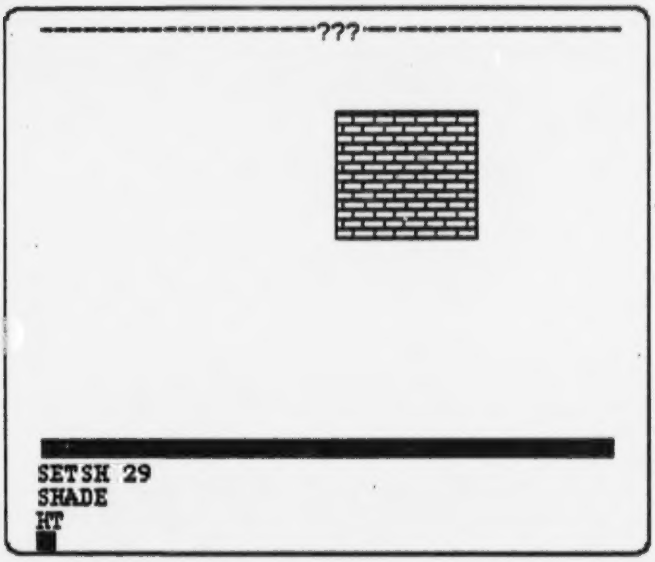
Then type

```
SHADE
```

and finally

```
HT
```

Now you see a square filled with bricks. Use the arrow keys to repeat this sequence, but use a different turtle shape. The square will fill with whatever shape you want.



```
SETSH 29
SHADE
HT
```

You need to remember the same steps when working with SHADE that you must use with FILL. Lift the pen when moving the turtle; put the pen down before shading; and move the turtle just a few steps if you have difficulty.

FILL and SHADE can be used to create interesting effects in drawings. Spend some time experimenting with them.

TIPS AND TECHNIQUES

If either FILL or SHADE does not work properly, it is most likely because you don't have the pen down or the turtle's pen is over a dot. First, type PD and try again. If that doesn't work, then the turtle is positioned so that LogoWriter thinks the area is already filled or shaded. Lift the pen. Move the turtle a small amount. Put the pen down and try again.

After shading, you most likely want to hide or move the turtle. The original turtle shape usually distorts the pattern created by using SHADE.

You can usually FILL an area after you SHADE it, but you can't SHADE after you FILL.

The behavior of FILL and SHADE differs slightly on different kinds of computers. Don't be surprised if you note those differences when working on different machines.

Since shading and filling areas make major changes on your page, it is a good idea to save your page before you use SHADE or FILL. Suppose you name your page MY.PAGE and press Esc to save it. Next you Fill an area. The FILL doesn't work right and you would probably like to begin again. First, type

```
NP "JUNK
```

or some other meaningless name. Press Esc. Then get the page MY.PAGE from the Contents Page and try again.

PROJECT SUGGESTIONS

Create a number of closed polygons on the screen. Use Turtle Move to place the turtle before drawing each design. Then again use Turtle Move to get the turtle inside each design to fill it. Can you create some interesting artistic effects using filled areas? What happens when the polygons overlap?

Use SHADE to create patterns that are part of a greeting card. For example, you might place lines so that the edges of the page are filled with a shape of your choice and then place a message in the middle of the card. You might even want to design a turtle shape of your own to personalize the card.

Design a sign announcing some school activity. Use a combination of filling, shading, and labels to make your sign.

Use the LogoWriter techniques that you have learned so far to create the cover for a report. You could create a turtle shape appropriate to the subject of the report. Remember you can stamp shapes as well as use them to fill areas.

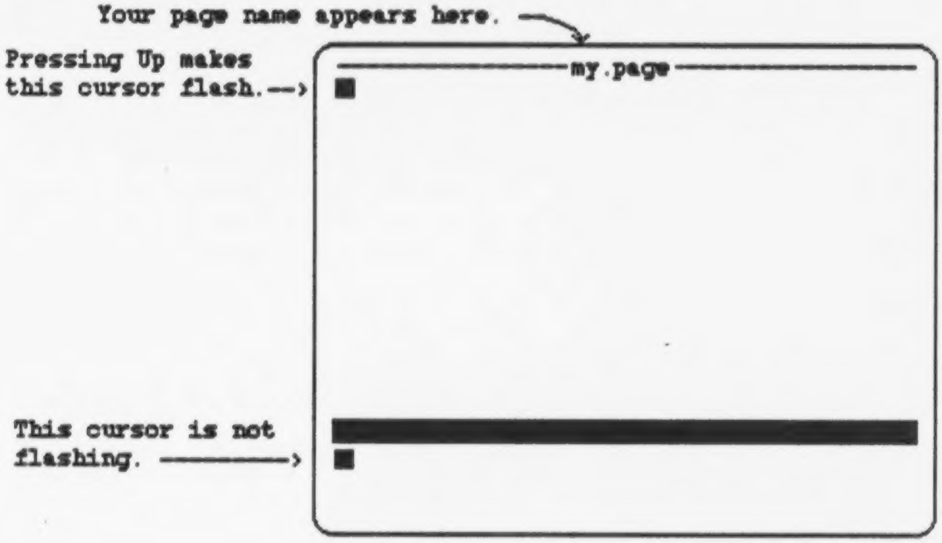
7. Using Text

So far you have been using only graphics on the Logo-Writer page. The text in the Command Center at the bottom of the screen is not part of the page. Recall that with Label, the letters are part of the graphics; they disappear when you type CG. In this chapter you will learn to put letters and numbers on the page that are not part of the graphics.

Start LogoWriter and get a new page. Name your page by typing

```
NAMEPAGE "page.name.you.choose
```

and then press the Up keys. (Check Appendix 8 for the keys to use for your particular machine.) Notice that the cursor is now flashing at the top of the page.

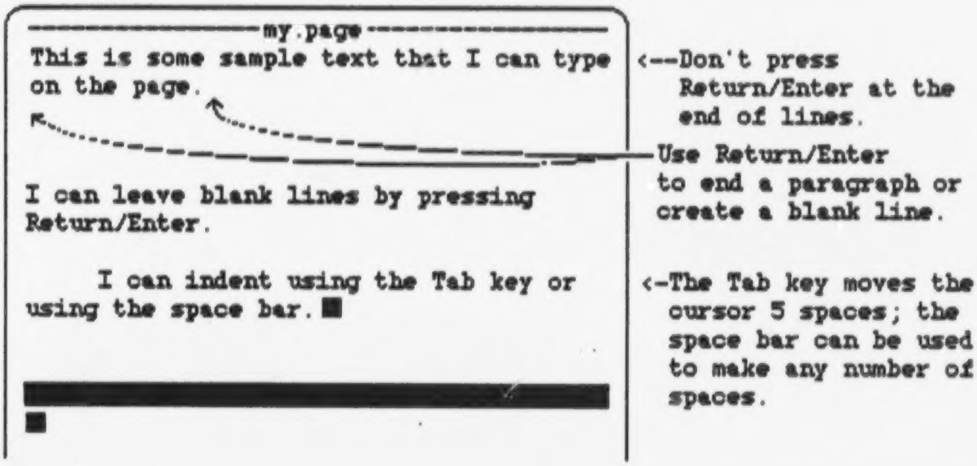


Your page name appears here.

Pressing Up makes this cursor flash.

This cursor is not flashing.

Begin typing. The words appear on the page instead of in the Command Center. Continue typing. Do not press Return/Enter when you get to the end of a line.



This is some sample text that I can type on the page.

I can leave blank lines by pressing Return/Enter.

I can indent using the Tab key or using the space bar.

<--Don't press Return/Enter at the end of lines.

Use Return/Enter to end a paragraph or create a blank line.

<-The Tab key moves the cursor 5 spaces; the space bar can be used to make any number of spaces.

Complete words automatically jump to the next line. This is called **word wrap.** Words on the page in text mode behave just like text in a word processor. In fact, you are now using LogoWriter's built-in word processor.

Now that you have some text on the page, press the Down keys (again check Appendix 8) to activate the cursor in the Command Center. Next type

```
CG
```

Nothing happens. That is because you now have words in text mode instead of graphics mode. To clear the text from the page, you must type

```
CT
```

for Clear Text.

To explore how the LogoWriter word processor works, clear the page (CT), hide the turtle (HT), and type the text exactly as shown in the diagram at the bottom of the page. Use the space bar and Tab key to indent the lines in the second paragraph.

There are several ways to print text. With the text shown below on the page, type

```
PRINTSCREEN
```

and the large letters that you have seen before with Label appear. Next type

```
PRINTTEXT
```

This time the words appear on the printer in the same places that they are on the page, but in regular sized type. In addition, there are margins at the left, top, and bottom of the page. If you want double spacing, you can use DSPACE. Use SSPACE to get back to single spacing.

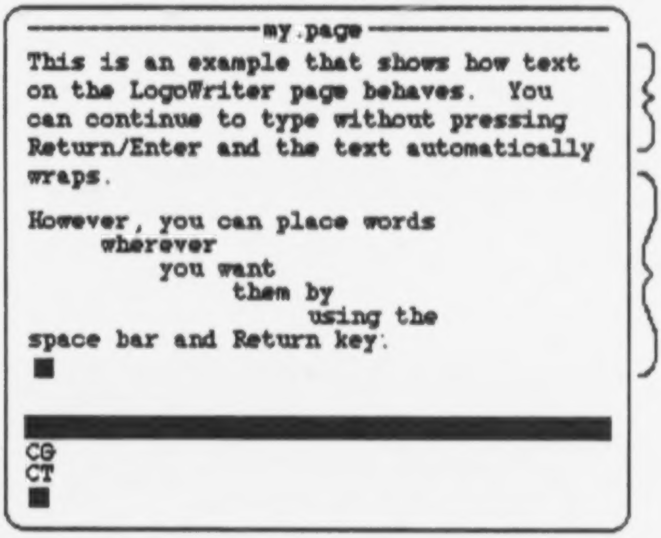
> Note: If you are using LogoWriter version 1.0, there are no margins on the page. If you are using LogoWriter version 1.1, you will need to type a SSPACE before typing PRINTTEXT. If you are using IBM Version 1.1, you must use DOUBLESPACE instead of DSPACE; SINGLESPACE instead of SSPACE.

Finally, type

```
PRINTTEXT80
```

and the text is printed using 60 columns (unless you pressed Return/Enter after each line) with margins at the left, right, top, and bottom of the page.

> Note: PRINTTEXT80 is set up for 10 characters per inch (Pica type). If your printer is set for some other type style, the text may not go all of the way across the page. Sometimes turning your printer off and then on again will reset your printer to 10 characters per inch.



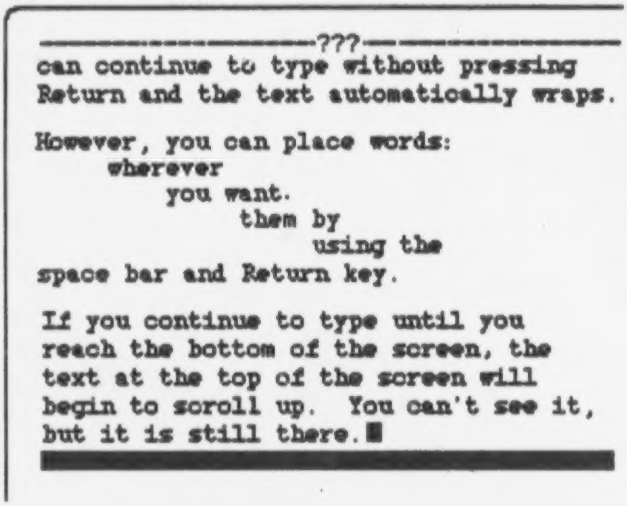
```
------------------my page------------------
This is an example that shows how text
on the LogoWriter page behaves.  You
can continue to type without pressing
Return/Enter and the text automatically
wraps.

However, you can place words
     wherever
         you want
              them by
                   using the
space bar and Return key.
■

CG
CT
■
```

Do not press Return/Enter after these lines.

Press Return/Enter after each of these lines.

Take a few moments to be sure that you understand how each of these commands for printing text works. Note that PRINTSCREEN prints all of the text *and* graphics that appear on the page. PRINTTEXT and PRINTTEXT80 print only text.

Everything you have created so far using LogoWriter has been visible on the page. Continue working with the page of text you created above. Place the cursor after the last word, and begin to type. Type whatever you want. Press Return/Enter whenever you want. What happens when you get to the bottom of the page? Did your screen scroll like this?



```
---------------------???---------------------
can continue to type without pressing
Return and the text automatically wraps.

However, you can place words:
     wherever
         you want.
              them by
                   using the
space bar and Return key.

If you continue to type until you
reach the bottom of the screen, the
text at the top of the screen will
begin to scroll up.  You can't see it,
but it is still there.█
```

There are a number of key combinations that help you move around on the page. They are listed below. Look at the summary in Appendix 8 to find what keys to use for your particular type of machine. (Not all of these work for every machine on which you can use LogoWriter.) Experiment with each of the following that works on your computer until you are comfortable using it. You might even want to write which keys to use on your machine next to each item in the list below.

Top of Page
Bottom of Page
Previous Screen
Next Screen
Beginning of Line
End of Line

(Note that Beginning of Line and End of Line move the cursor from one Return/Enter character to the next.)

It is much easier to change what you have written on the page using the LogoWriter word processor than it is when you use Label. You can add things, delete things, and correct things with ease. That is why you should Label *only* for short graphics signs and labels.

With some text on the page, try this. Move the cursor to the middle of a line and start to type. Notice how the words you type are automatically **inserted**. That is, they move out of the way as you type.

Next, try pressing the Delete or Backspace key. Notice that this key causes letters to be erased from right to left. When you want to erase something, you can place the cursor *after* it and then press Delete/Backspace. Experiment with making changes in the text on the page. (You can delete the character under the cursor using Control/D if you are using an Apple computer.)

When you are deleting text, some apparently strange things can occur:

- Sometimes words become "glued" together. Press the space bar to separate them.
- Sometimes words jump from one line to the next. This is because you have deleted the invisible space between the words.
- If you put the cursor on a blank line and press Delete, the blank line will disappear. This is because you have deleted the invisible "carriage return" character. Simply press Return/Enter to replace it.
- If you put the cursor on the first character of a paragraph and press Delete/Backspace, the paragraph moves up to join the one before it. Again, the invisible carriage return character was deleted. Press Return/Enter to put it back.

Note: If you are familiar with word processing, you may want to learn more of the capabilities of the LogoWriter word processor. If so, see Chapters 13, 14, and 22. Keep in mind, however, that LogoWriter includes only an entry level word processor. You will not find many of the capabilities that are included in today's powerful software that is devoted entirely to word processing.

If you are using the GS version of LogoWriter, you can change the color of the text (and the cursor) on the page using the command SETTC. Typing

```
SETTC 2
```

changes the text and the text cursor to color number 2. If you want to know the color number of the text under the text cursor, you can type

```
SHOW TC
```

TC reports the the text color number.

TIPS AND TECHNIQUES

If you have experience with other word processors, you may wonder how you can accomplish such tasks as centering text, setting margins, or underlining words. None of these features are available in the LogoWriter word processor. However, the LogoWriter word processor has its own advantages. It is very easy to use. There are no elaborate menus or key combinations to learn. In addition, you will soon learn that you can easily insert graphics into your text. If you want to do simple desk top publishing, then LogoWriter is an excellent tool. If you want to write long papers, you probably want to use another word processor. Don't try to make the LogoWriter word processor meet all of your word processing needs.

Text created in some other word processors can be "imported" into LogoWriter, and text created in LogoWriter can be "exported" to other word processors. To see if you can exchange text between your version of LogoWriter and your word processor, see Appendix 7.

If you type CT, all of the text on the page disappears, including that which has scrolled off the page. If you accidentally type CT, it is possible to get your text back. *Before you press any other keys*, type

```
UNDO
```

in the Command Center and your text will reappear. (Note: UNDO is not available in LogoWriter Versions 1.0 and 1.1).

PROJECT SUGGESTIONS

Now that you know how to place text on the screen, you can use this capability to do a variety of things.

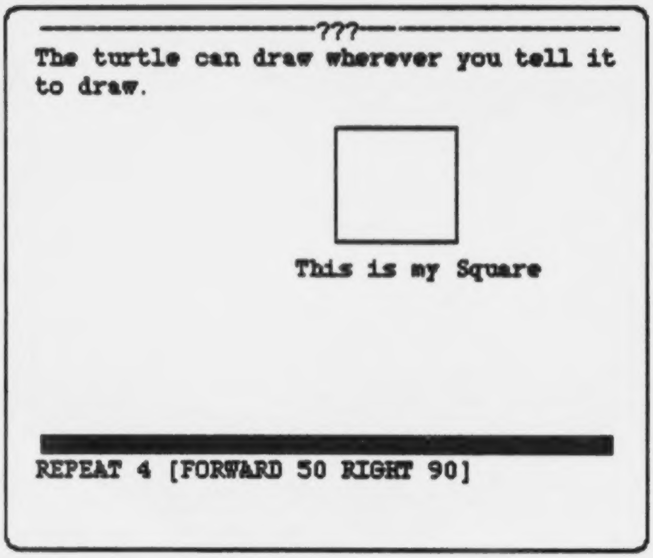
- Make a sign (printed using PRINTSCREEN).
- Write a letter to a friend.
- Do a short report for another class.
- Write a poem.
- Make a design or picture using letters, numbers and other keyboard symbols. (Pictures such as this used to be called "typewriter pictures.")

8. Mixing Text and Graphics

You have now learned how to put text on the page. Earlier you learned how to put graphics on the page using the turtle and Label. Using LogoWriter, you can combine graphics and text on the same page. Get a new page and try this:

```
REPEAT 4 [FORWARD 50 RIGHT 90]
```

Then, use Label to write "This is my Square" under the square. Now use the Up keys and type "The turtle can draw whatever you tell it to draw." Your page should look like this:



```
The turtle can draw wherever you tell it
to draw.

This is my Square

REPEAT 4 [FORWARD 50 RIGHT 90]
```

Next, press Esc to save the page. Get the page and again type

```
CG
```

What happens? Do you see why?

Next type

```
RESTORE
```

to get back the page you saved. Now type

```
CT
```

What happens? Do you see why?

When you are working on a project that contains both text and graphics, you can correct the parts separately, or even start one part over without erasing the other. This can be very handy when working on a more complex project.

TIPS AND TECHNIQUES

When you have put a lot of work into a LogoWriter page, it is very frustrating to make a major error or accidentally erase your work. You have learned a number of techniques for protecting your work. When you make a mistake, *don't panic.* Before you press any keys, take a moment to think.

Can you use UNDO to restore lost text? Immediately go to the Command Center and type UNDO.

Have you recently saved the page? If so, type RESTORE in the Command Center to get back a recent version of your work.

Have you been saving your pages on a regular basis using names like MY.PAGE1, MY.PAGE2, MY.PAGE3, etc. If so, you can go to the Contents Page and get the most recent copy of your page.

If none of these techniques work this time, in the future, make it a habit to save your work every 10 - 15 minutes. Better yet, save it every time you complete a major change on your page. The time taken now to save your work will pay off later if you make a mistake.

PROJECT SUGGESTIONS

Use a mixture of text and graphics to create pages of your choice. You might want to do a greeting card, a sign, or a page for a report. Remember that you can create any shape you want using the Shapes page. Try extending some of the suggestions made at the end of Chapter 7 by adding graphics to them.

9. Writing Procedures

All of the pages you have created so far have been done either by typing commands in the Command Center, or by putting text directly on the page using the LogoWriter word processor. When you saved, everything you created was put on your disk exactly as it appeared on the page.

Have you ever created a neat graphics design that you would like to use on another page? Perhaps you remembered the commands you used to make it, perhaps not. Without a list of the commands, it can be hard to recreate a graphics design. Your have already seen that the commands in the Command Center are not saved. If you start LogoWriter at another time and get your page, the commands you used to create it are not there.

There is a way to save commands that you have used to create pages in Logo. Recall that you could flip the Shapes page to see magnified copies of the shapes. You can also flip the page on which you put text and graphics. Try it. Use the Flip keys (See Appendix 8) to see the back of the page. Notice that it says "Flip side" at the top of the screen and that the Command Center is still visible. Compare your screen with the diagram at the bottom of the page.

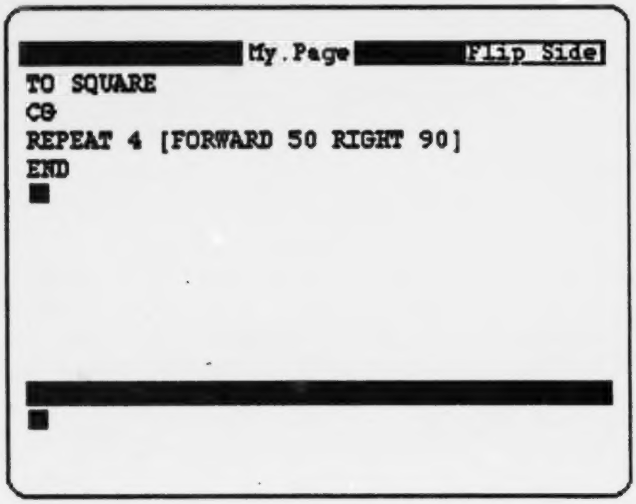
Use Flip to flip back to the front of the page. Notice the location of the flashing cursor. Where is it when you are on the front of the page? Where is it when you are on the back of the page?

Now try this. Start on the front of the page. Flip the page and use the Down keys to get to the Command Center. Type

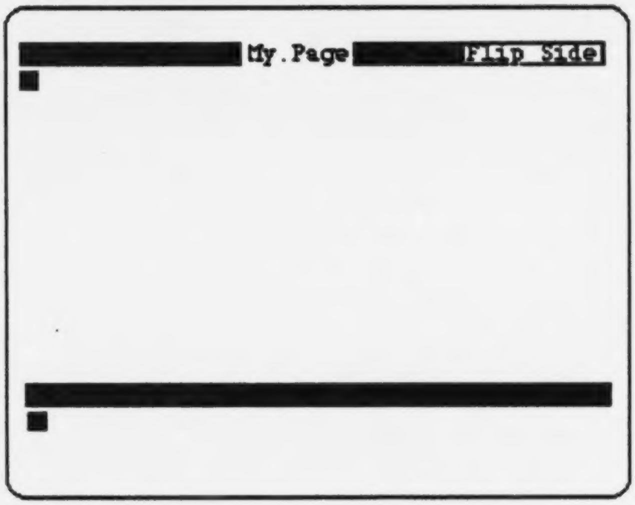
```
REPEAT 4 [FORWARD 50 RIGHT 90]
```

What happens?

Flip to the back of the page and type the lines shown below



```
TO SQUARE
CG
REPEAT 4 [FORWARD 50 RIGHT 90]
END
```



This cursor is flashing ——>

This cursor is NOT flashing —>

Flip to the front of the page and type

```
SQUARE
```

What happens? You have just written your first Logo procedure.

A procedure is a collection of Logo commands. Procedures must begin with the word TO and end with the word END. The structure of a procedure is as follows:

```
TO name.of.procedure    <—title line

  Commands go here      <—body of the procedure

END
```

Procedures must be written on the back of the page. They are then saved with the page.

In most versions of Logo there is a separate "editor" where you write procedures. This editor is usually a low-level text processor that requires you to learn a set of editing keys, often different from those used in the "immediate mode" where commands are typed one line at a time. The LogoWriter "editor" involves using the word processor on the Flip side of the page. When writing procedures in LogoWriter you use the same editing keys that work in the LogoWriter word processor and in the Command Center.

While the mechanics of procedure writing are rather simple, the idea embodied in procedure writing is profound. When you create a series of instructions to complete a particular task and then put them into a named procedure you have "taught" Logo a new word. This new word behaves exactly as if it were a primitive. If you don't look on the flip side of the page, you have no way of knowing whether the commands that you type in the Command Center consist of primitives or user-defined procedures. The fact that you can easily create procedures that function exactly like primitives is what makes Logo an easily **extensible** language.

The flip side of the LogoWriter page can contain as many procedures as the memory of the computer and/or the version of LogoWriter you are using allows. Each of these procedures must have a unique name. Each new name becomes part of the LogoWriter vocabulary available when you use the page.

Since all of the commands that you have learned so far can be used in a procedure as easily as they can be used in the Command Center, you now have the tools to write procedures to do many interesting things.

You can move the turtle around.

```
TO PLACE.TURTLE
CG
PU
FORWARD 20
RIGHT 60
FORWARD 30
PD
END
```

You can change the shape of the turtle and the color of its pen.

```
TO CHANGE.TURTLE
ST
SETSH 25
SETC 3
END
```

You can stamp turtle shapes.

```
TO STAMP.IT
SETSH 15
PD
STAMP
PU
FORWARD 20
END
```

Take some time now to experiment with writing a procedure. Notice as you experiment how powerful this idea of procedure writing is. Now *you* are teaching Logo how to do new things. The computer is now under your control.

TIPS AND TECHNIQUES

When you begin using the Flip side of the page to write procedures, it is easy to get lost. Learn to look at the screen carefully.

- Are you on the front of the page or the Flip side? (You see "Flip Side" if you are not on the front.)
- Where is the cursor flashing?
  - Not at all? You are in Turtle Move or Label mode.
  - At the bottom of the screen? You are in the Command Center.
  - At the top of the screen, no Flip Side? You are in the word processor.
  - At the top of the screen; it says Flip Side? You are ready to write procedures.

You can use the Up and Down keys on either side of the page. That is, you can use Down to get to the Command Center when you are writing procedures. Be especially careful if you do this. If you accidentally type CT, you can erase all of your procedures. (Remember, you can use UNDO.)

Don't forget the END on procedures. Leaving out the END statement will confuse LogoWriter. It won't recognize the next procedure in your list. It is helpful to put a blank line between procedures. Then you can more easily see where one procedure ends and the next begins.

PROJECT SUGGESTIONS

Write a procedure to make a small design. Flip to the front side of the page, and then use a combination of Turtle Move and your procedure to place your design in various places on the page.

Write a procedure to stamp two different shapes on the page. Flip to the front side of the page and use a combination of Repeat, Turtle Move, and your procedure to make a border of shapes on the page.

Write a procedure to make an elaborate graphics design using the turtle. Flip to the front side, use your procedure to make the design, and then use the word processor (press the Up keys) to add a description of your design on the screen.

Create a shape of your own and then write a procedure to stamp your shape in a pattern on the page. Add some text using Label. (In the next chapter you will learn how to put LABEL into a procedure.)

Write a procedure that draws a graphics design and changes the pen color randomly with each line it draws. You might use something like the fluffy ball created in Chapter 2. Use Label to put a title on your design.

10. More Than One Procedure

You now know how to write a procedure using the flip side of the page. As mentioned in the last chapter, you can write as many procedures as you want on the Flip side of the page, as long as there is space in the memory of the computer.

Carefully follow the example given below. It shows you how to write a number of procedures and put them together into a program.

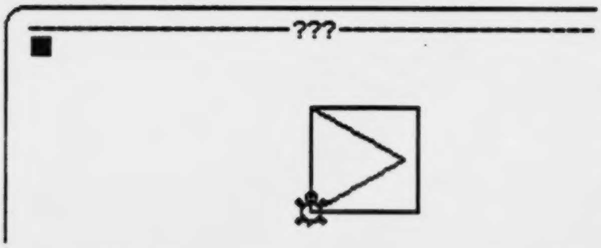
Suppose you wanted to have the turtle draw a house on the page and then put a "For Sale" sign on it. You can, of course, do this from the Command Center. However, if you do it with procedures, then you can easily make a number of identical houses.

To begin, notice that you can draw a simple house made up of a square for the main part of the house and a triangle for the roof. Flip the page and write these procedures for drawing a square and a triangle.

```
TO SQUARE
REPEAT 4 [FORWARD 40 RIGHT 90]
END

TO TRIANGLE
REPEAT 3 [FORWARD 40 RIGHT 120]
END
```

Flip the page back to the front. Type

```
SQUARE
TRIANGLE
```



You see the parts of the house...but they are not in the right place. Type CG and see if you can figure out commands to get the roof in the right place. These commands should become part of a procedure to get to the roof. One solution might be

```
TO GET.TO.ROOF
FORWARD 40
RIGHT 30
END
```

Now flip the page and type

```
CG
HT
SQUARE
GET.TO.ROOF
TRIANGLE
```

Does the house look the way you want it to? If not, modify the procedures until you are satisfied with your house. These procedures you create can become part of a HOUSE procedure:

```
TO HOUSE
SET.UP
SQUARE
GET.TO.ROOF
TRIANGLE
END

TO SET.UP
CG
HT
END
```

Now, you can type HOUSE from the Command Center and see your house. Notice that a SET.UP procedure was added to prepare the page for the house drawing.

Next, you need to work on the "For Sale" sign. This time you can't use Label because you want the sign to be created in a procedure. That is possible using the LABEL command. From the Command Center, type

```
CG
LABEL [HELLO]
```

Notice that the word "Hello" appears next to the turtle. You can type HT to see the word better. Remember how you corrected errors in Label mode by typing over them? The same thing happens when you use LABEL as a command in the Command Center. Move the cursor up to the

```
LABEL [HELLO]
```

line in the Command Center and press Return/Enter. Do this several times. What happens?

Using this technique, you can make flashing signs. Try typing

```
REPEAT 10 [LABEL [HELLO] WAIT 10]
```

WAIT 10 causes LogoWriter to pause for 10/20ths or 1/2 second. Try changing the length of the pause until you have a sign that flashes at a speed that you like.

Now you can make a flashing "For Sale" sign for the house. Flip the page and type

```
TO FLASH.SIGN
REPEAT 9 [LABEL [For Sale] WAIT 20]
END
```

The sign isn't in the best of places. You need to write a procedure to move the turtle before placing the sign. For example, you might write

```
TO PUT.SIGN
PU
RIGHT 60
FORWARD 50
PD
HT
END
```

However, there is a neat trick for converting using the Label keys into commands that you can use in a procedure. First, use your procedure to draw your house. Type

```
HOUSE
```

Next, use the Turtle Move keys to place the turtle where you want the sign to appear. Press Escape to leave Turtle Move. In the Command Center type

```
SHOW POS
```

Two numbers surrounded by square brackets appear. These are the **coordinates** of the turtle. (POS reports the current location of the turtle in Cartesian Coordinates. See Chapter 36 for more details.) Suppose you see

```
[60 25]
```

in the Command Center. Flip the page and write the following procedure:

```
TO PLACE.SIGN
PU
SETPOS [60 25]
PD
END
```

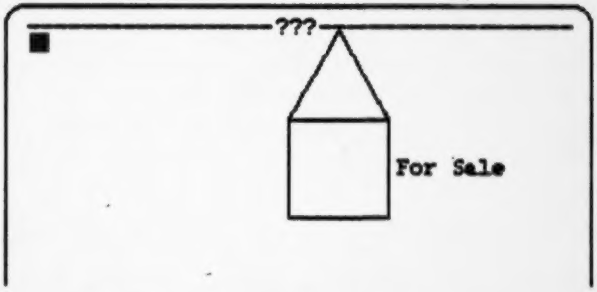
SETPOS sets the position of the turtle to a location given in the coordinate list that follows it. Now type

```
HOUSE
PLACE.SIGN
FLASH.SIGN
```

The house appears with its flashing sign..Now you can put this all together in one procedure. Flip the page, move the cursor to the top of the page and type

```
TO HOUSE.AND.SIGN
HOUSE
PLACE.SIGN
FLASH.SIGN
END
```

Now, typing HOUSE.AND.SIGN in the Command Center gives you this picture, complete with flashing sign!

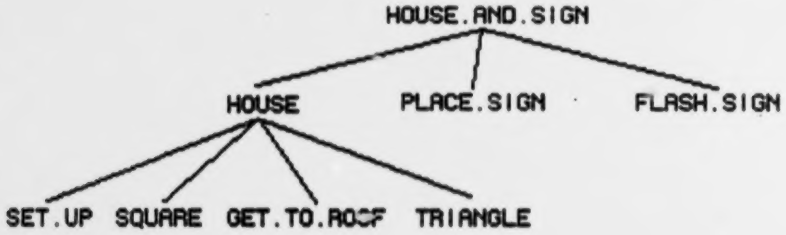


When you are writing a program with a number of procedures, it is a good idea to include a description of the program. On the flip side of the page, move the cursor to the top and type a description. You might type something like

> <<The following program draws a house with a flashing "For Sale" sign. Type HOUSE.AND.SIGN in order to run the program.>>

The "<<" symbols aren't necessary. They just set off the remark from the procedures themselves.

The procedure HOUSE.AND.SIGN is the **main procedure, or top level procedure.** It uses a number of other procedures. These procedures are called **subprocedures** of HOUSE.AND.SIGN. A good way to visualize the relationships among procedures is by using a **procedure tree.** In a procedure tree, a subprocedure is drawn below the procedure that uses it. For example, the procedure tree for HOUSE.AND.SIGN looks like this:



Notice that the HOUSE procedure, which is a subprocedure of HOUSE.AND.SIGN, has four subprocedures of its own.

The word **superprocedure** is also commonly used to describe relationships among procedures. For example, HOUSE.AND.SIGN is a superprocedure of FLASH.SIGN.

To summarize:
- HOUSE.AND.SIGN is the main, or top level procedure.
- HOUSE, PLACE.SIGN, and FLASH.SIGN are subprocedures.
- HOUSE.AND.SIGN is a superprocedure of HOUSE, PLACE.SIGN, and FLASH.SIGN.
- SQUARE, GET.TO.ROOF, and TRIANGLE are subprocedures of HOUSE.
- HOUSE is a superprocedure of SQUARE, GET.TO.ROOF, and TRIANGLE.

Notice how much easier it is to see the relationships among procedures using a procedure tree. The list of the relationships is hard to follow. The procedure tree gives you a visual image of the structure of your program. You should always draw a procedure tree when you are working with more than 2 or 3 procedures.

There are some "rules of thumb" that you should remember when you are drawing a procedure tree.

1. The name of the program (the procedure name you type to run the program) is at the top of the procedure tree on a line by itself.
2. Each subprocedure of the main procedure is listed on the next level (or line). These subprocedures are listed left to right as they occur in the main program. That is, the first subprocedure is listed on the left; the next subprocedure used to the right of the first, and so forth.
3. A line connects each procedure with its subprocedure(s).
4. Each procedure in the program along with its subprocedures is represented using the rules given in step 2.
5. If a procedure is a subprocedure of more than one procedure, it should appear "under" each procedure from which it is called.
6. If a procedure "calls itself" (Chapter 19) then the name of the procedure appears below the name of the procedure connected by a line just like any other subprocedure.
7. All "user defined" procedures (those that you write) should appear in the procedure tree.
8. No primitives (such as CG, FORWARD, or REPEAT) are listed in the procedure tree.

When you have completed your procedure tree, you should be able to tell exactly what procedures are called from any procedure. You should be able to tell what procedures call any subprocedure. A carefully drawn procedure tree can be very helpful in solving any problems that you have with your program.

TIPS AND TECHNIQUES

Mistakes in computer programs are called "bugs." The process of removing these errors is called "debugging." You will quickly learn that you spend more time debugging your Logo programs than you do writing them. Don't let that discourage you. Even professional computer programmers spend most of their time debugging. You should consider the time you spend debugging part of the learning process.

The ideas in this section of each chapter have been designed to help you avoid or fix bugs when you were working in the immediate mode. Increasingly this section will include techniques for debugging. Read them carefully. They will save you a lot of time in your future work.

When you are working with a number of procedures, you should be sure that each procedure works by itself. That is, after writing any one procedure, flip the page and type the name of the procedure. As you build up your program, test each procedure as you go. Thus, in the above example, after you tested SQUARE, TRIANGLE, and GET.TO.ROOF separately, you should test HOUSE to be sure all of the pieces work together properly. Don't try to write your entire program all at once and then test it by typing the name of the top level procedure. There are simply too many things that can go wrong in a program that is more than 2 or 3 procedures long.

Give your procedures meaningful names and keep your procedures quite short. It is easy to find yourself adding more and more commands to a procedure. Soon you have a very long procedure that is hard to debug. Perhaps the name of the procedure no longer describes the function of the procedure. Take time to subdivide and rename procedures as you work. It will save a lot of time later when you are debugging.

PROJECT SUGGESTION

Now that you know how to write multiple procedures, plan a picture of your choice and write the procedures to create it. Move along step by step like the example in this chapter. Draw a procedure tree as you go. Be sure to keep the individual procedures short. You should not have any procedures that are longer than 20 lines with one statement on each line. If you put too much into a procedure, it becomes hard to find errors.

11. A Word About Designing Programs

There are a number of ways to go about creating programs. Different people prefer different approaches. Different computer languages lend themselves to different methods. Now that you are beginning to write programs containing a number of procedures, you should spend some time thinking about the ways to design a programs. You should be examining your own programming to see which method(s) suit your particular learning style.

TOP DOWN PROGRAMMING

Top down programming refers to writing a program "from the top." Think about the HOUSE.AND.SIGN program from the previous chapter. If you use a top down approach, the first procedure you write would be

```
TO HOUSE.AND.SIGN
HOUSE
PLACE.SIGN
FLASH.SIGN
END
```

Next you would write the HOUSE procedure:

```
TO HOUSE
SET.UP
SQUARE
GET.TO.ROOF
TRIANGLE
END
```

Then you would write SET.UP, SQUARE, GET.TO.ROOF, and TRIANGLE. Finally, you would write PLACE.SIGN and FLASH.SIGN.

With this method, you plan the whole and then write ever smaller parts. This is a very regimented and structured approach to problem solving. You must do a lot of planning in advance. You need to know exactly what you want to accomplish before you even begin. This approach allows little room for exploration as you go.

BOTTOM UP PROGRAMMING

In bottom up programming, you start with small pieces and build them up into a complete program. If you used bottom up programming in the HOUSE.AND.SIGN program, you wouldn't start with a plan. You might first create the SQUARE procedure and then the TRIANGLE procedure. After some fiddling with these, you might discover that, when put together, they make a nice house. That might lead to the HOUSE procedure.

You might be experimenting with the LABEL command and get the idea to write a procedure to put a sign next to the house. Next would come the FLASH.SIGN, and then the PLACE.SIGN procedure. Last of all, the HOUSE.AND.SIGN procedure would be written.

With this method, there is little advance planning. You play with an idea, turn it into a procedure, and let that idea lead you to another. You have little or no idea where you are going as you begin. Logo lends itself to the bottom up style of programming better than most programming languages. Languages like Pascal all but force programming in a top down style.

Much of what is written about Logo implies that the only "right" way to program in Logo is bottom up. This emphasis on bottom up style grows out of the philosophy of discovery learning that is closely allied with Logo. It is assumed that the Logo programmer will experiment with an idea and then use that idea as part of a larger project. However, in the first part of this chapter, you saw how one might write a program using a top down approach to problem solving in Logo.

MIDDLE OUT PROGRAMMING

Very few people actually program *either* in a true top down or bottom up style. Most people use some combination of methods for writing programs, which might be described as "middle out." Middle out is the method that was actually used for writing the HOUSE.AND.SIGN program. There was an initial idea of what the program should do. Then small parts were written. Finally all of the pieces were put together. Some things were done top down, other things bottom up, but most things were done from the middle, working both towards the top and the bottom.

POLISHING YOUR PROGRAM

It doesn't really matter what programming style is used to do a project. However, it is best if the final program looks structured and well organized, much as if it had been done in a top down manner. That is, there should be a top level procedure with a title that describes the entire program. This top level procedure should be made up primarily of procedure calls whose titles describe the sub parts of the program. Many of the subprocedures in the program may also be made up of procedure calls that describe even smaller parts of the program.

Both the programming process *and* the final product are important in any programming project. During the development of a project, you probably focus on writing the code and on debugging the results. (Recall that things in programs that don't work the way you want are called "bugs.") However, when you have finished a program in Logo, it should be polished for "publication," just as you polish your final draft of a report before you turn it in to a teacher or employer.

What is the "correct" style to use when writing a Logo procedure? Are there "rules" that you should follow? The issue of programming style is one for which there is very little agreement within the Logo community. At one end of the spectrum are those that feel that to impose *any* rules for writing procedures is inappropriate. They feel that such rules will interfere with creativity and exploration. This book uses a very consistent and somewhat rigid style. This approach lies nearer the other end of the spectrum.

Any Logo programs that you write should be polished for "publication" no matter who is going to see them. Polishing the final product makes it more readable both to yourself and to others. You will be surprised how quickly you can forget the details of a program that you spent many hours writing. Only weeks later a poorly written program can be very hard to untangle.

Here are some guidelines to help you begin developing a readable programming style.

1. Procedures should have meaningful names. That is, the name of the procedure should describe *exactly* what happens in that procedure.

2. There should be no more than one instruction per line.

3. No procedure should be longer than 20 lines (approximately the number of lines that fit on the page at one time).

4. The main or top level procedure should contain only user defined procedure names. That is, you should not use a LogoWriter primitive in the main procedure.

5. If an instruction is longer than one line, you should indent the second and subsequent parts of the line in a logical and readable manner.

6. When you use procedure inputs or global names, you should use meaningful names that describe the function of that name. (Inputs and variables will be discussed later in this book.)

These rules are perhaps unnecessarily rigid for your long-term work with Logo programming. However, they are designed to give you specific guidelines to follow from the outset that will help you in debugging your future, more complex programs. As you continue your work with Logo beyond this book, you will no doubt relax or modify these rules in ways that are comfortable for you.

Programs written using the above guidelines are much easier to work with. Other people can easily tell what each part does. You can more easily make changes in your own work at a later date. Most importantly though, is the way in which a program written in this style is easy to debug. Sometimes the bugs are easy to find, sometimes they aren't. If the program is written well, then the debugging process takes less time.

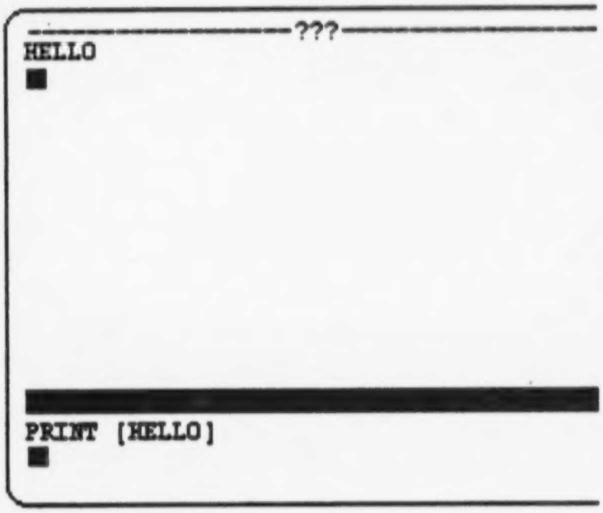
PROJECT SUGGESTION

Select a project that involves writing a number of procedures. As you are working on it, examine your own programming style. Which of the styles mentioned above best describes the way you prefer writing programs? Be sure your final program is polished for publication.

12. Using PRINT

You have always put text directly on the front of the page using the LogoWriter word processor . There is a command in LogoWriter for putting text on the page as well. Be sure you are on the front of the page and type

```
PRINT [HELLO]
```



```
HELLO
```

```
PRINT [HELLO]
```

The word "HELLO" appears at the top of the page. You can also use PRINT from a procedure. Flip the page and type

```
TO GREET
PRINT [Greetings, Mary!]
END
```

Now, flip the page and type

```
GREET
```

Then,

```
Greetings, Mary!
```

appears on the page.

Each time you issue a PRINT command, LogoWriter prints what follows and then moves the cursor to the next line. Thus you can write a procedure to create a more elaborate greeting as follows:

```
TO GREET
CT
PRINT [Greetings, Mary!]
PRINT [How are you?]
PRINT [I'm fine!]
END
```

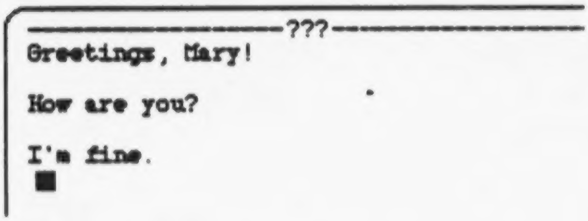
When you run this procedure,

```
Greetings, Mary!
How are you?
I'm fine!
```

appears on the page.

What if you want some blank lines between the parts of your greeting? You can add blank lines by printing an empty line.

```
TO GREET
CT
PRINT [Greetings, Mary!]
PRINT []
PRINT [How are you?]
PRINT []
PRINT [I'm fine!]
END
```



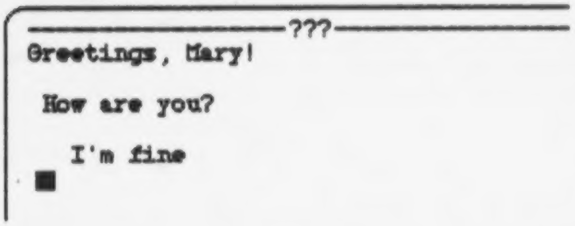
```
Greetings, Mary!

How are you?

I'm fine.
```

Then your message will be double spaced.

What if you want to put some spaces in front of some of the lines? You can do this by inserting blanks. There are several ways to do this. One is to use the backslash (\) to quote a blank space. (See Chapter 17.) Another is to use CHAR 32. CHAR 32 is simply a way to represent a space. Thus, you might enter

```
TO GREET
CT
PRINT [Greetings, Mary!]
PRINT []
(PRINT CHAR 32 [How are you?])
PRINT []
(PRINT CHAR 32 CHAR 32 [I am fine!])
END
```

If you type GREET, you now see



```
Greetings, Mary!

 How are you?

   I'm fine
■
```

Did you notice the parentheses ( ) around the last two PRINT statements? These allow PRINT to have more than one input. In this case both CHAR 32 *and* the list to print are inputs to PRINT. You can have as many inputs to PRINT as you want by using parentheses in this way.

Inserting spaces by adding a lot of CHAR 32's in the PRINT line is a bit cumbersome. Here is a "tool" procedure that you can use. (Perhaps you'll want to keep it on a separate page for use later.) You'll learn more about writing procedures like this in Chapter 26.

```
TO SPACES  :NUMBER
REPEAT :NUMBER [INSERT CHAR 32]
END
```

Now, the GREET procedure becomes

```
TO GREET
CT
PRINT [Greetings, Mary!]
PRINT []
SPACES 1
PRINT [How are you?]
PRINT []
SPACES 2
PRINT [I'm fine!]
END
```

TIPS AND TECHNIQUES

When writing a procedure using PRINT, be sure you are on the front of the page before you run it. If you use the Down keys to go to the Command Center instead of flipping the page, the text after PRINT will appear on the Flip side. Text created by the PRINT statements in your procedure will be mixed with your procedures themselves.

If you know another programming language such as BASIC or Pascal, take special note of the use of parentheses with PRINT. The left parenthesis goes *before* the word PRINT, not after it.

PROJECT SUGGESTIONS

Write a procedure to put a poem spaced neatly on the screen. Make use of blank lines and leading spaces.

Write a procedure to paint a "word picture." It might be a pun, like

```
    MAN          (man overboard)
   BOARD
```

or something more descriptive like

```
FALLING D
         O
          W
           N
```

13. Printing Text on a Printer

You have probably printed your work in LogoWriter using PRINTSCREEN. The PRINTSCREEN command prints graphics and text exactly as they appear on the page. The letters are "double sized," unlike the kind of printing you usually get from a word processor. You have now used both the word processor mode and the PRINT command to put text on the page. In Chapter 7, you learned that you can print text in a normal size using the PRINTTEXT and PRINTTEXT80 commands. These commands work in the same way whether you put text on the page using the word processor or the PRINT command.

Recall that

```
PRINTTEXT
```

causes LogoWriter to print the text with the lines broken exactly as they appear on the screen, but indented one inch (10 characters) on the paper. You can also use

```
PRINTTEXT80
```

This prints the text with an inch-wide left margin and a 60 character line.

You can also change the spacing of text that you want to print. There are two LogoWriter primitives that allow you to change spacing.

```
SSPACE
```

causes text to be printed with no extra spaces between lines.

```
DSPACE
```

causes the text to be double spaced. Before you print anything, you should decide whether you want single or double spacing. It's a good idea to type either SSPACE or DSPACE in case the person who used LogoWriter before you changed the spacing.

Recall also that there are a few differences among versions of LogoWriter.

- In version 1.0, there are no margins on the page.
- In some versions of LogoWriter, you will need to type a SSPACE before typing PRINTTEXT in order to have single spaced text.
- PRINTTEXT80 is set up for 10 characters per inch (Pica type). If your printer is set for some other type style, the text may not go all of the way across the page. Sometimes turning your printer off and then on again will reset your printer to 10 characters per inch.
- In IBM Version 1.1, you must use DOUBLESPACE instead of DSPACE; SINGLESPACE instead of SSPACE.

TIPS AND TECHNIQUES

When you are printing procedures, you should always single space and use PRINTTEXT. Double spacing makes procedures harder to read and debug. PRINTTEXT shows you the procedures exactly as they appear on the screen.

PROJECT SUGGESTION

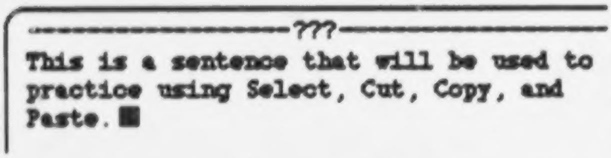
Use the word processing capabilities of LogoWriter to prepare a short paper for another course. Then try printing it in different ways: single and double spaced; using only 40 columns; and using the full width of the paper. You might even want to compare normal text with the larger text produced using PRINTSCREEN.

14. Cut, Copy, and Paste

In most word processors, you can "pick up" a piece of text from one place and either copy it or move it to another place. This allows you to easily rearrange things that you have written. With the LogoWriter word processor you can also move text from one place to another.

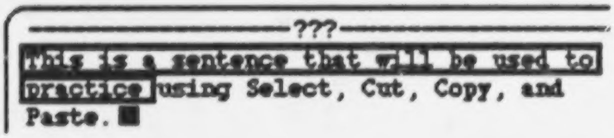
IN THE WORD PROCESSOR

After you start LogoWriter, press the Up keys to move the cursor onto the page. Then type the following text:



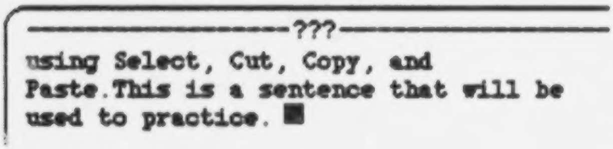
```
----------------???----------------
This is a sentence that will be used to
practice using Select, Cut, Copy, and
Paste. ■
```

Next, use Top of Page (See Appendix 8 for the keys to use on your computer) to move to the upper left hand corner of the page. Then press the Select keys. Nothing seems to happen. Now use the arrow key to move to the right. Characters that you pass over become the inverse of what they currently are. Use the left, right, up and down arrows to highlight the text that is outlined below:



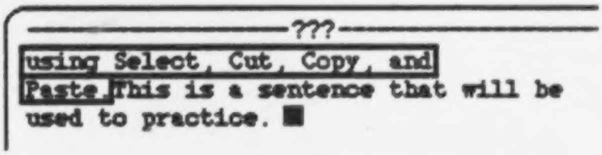
```
----------------???----------------
This is a sentence that will be used to
practice using Select, Cut, Copy, and
Paste. ■
```

Now press the Cut keys. (See Appendix 8 for the keys you need to use in this chapter.) The highlighted portion disappears. Move the cursor until it is after the word "Paste" in the above paragraph. Then press the Paste keys. The sentence now looks like this



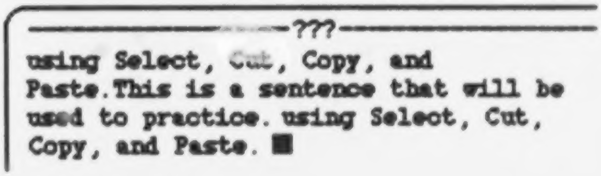
```
----------------???----------------
using Select, Cut, Copy, and
Paste.This is a sentence that will be
used to practice. ■
```

Next, highlight the following boxed section using Select and the arrow keys:



```
----------------???----------------
using Select, Cut, Copy, and
Paste.This is a sentence that will be
used to practice. ■
```

Press Copy. The highlighting disappears. Move the cursor until it is after the word "practice." Press Paste and the paragraph looks like this:



```
----------------???----------------
using Select, Cut, Copy, and
Paste.This is a sentence that will be
used to practice. using Select, Cut,
Copy, and Paste. ■
```

Take some time to experiment with using Select, Cut, Copy, and Paste. Don't worry if you have a few problems at first. If you get "lost" and aren't sure where you are in the text-moving process, press Esc and try again.

ON THE FLIP SIDE OF THE PAGE

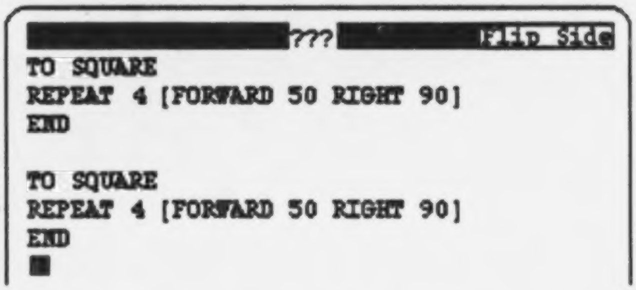
The text-moving commands work exactly the same way on the flip side of the page. Suppose you have the following procedure :

```
TO SQUARE
REPEAT 4 [FORWARD 50 RIGHT 90]
END
```

and you decide that you would like to have a procedure to draw a triangle. You can use Select, Copy, and Paste to save typing.

1. Put the cursor on the word TO.
2. Press Select.
3. Use the arrow keys to highlight the whole procedure.
4. Press Copy. The highlighting disappears.
5. Move the cursor to after the word END.
6. Press Paste.

You now have a second copy of the procedure–see the illustration at the top of the next page. (If the title line of the copy of the procedure is next to the END statement of the original procedure, simply move the cursor until it is after the first END and press Return/Enter a couple of times. In fact, it is generally a good idea to leave a blank line between procedures. It makes your program easier to read.)

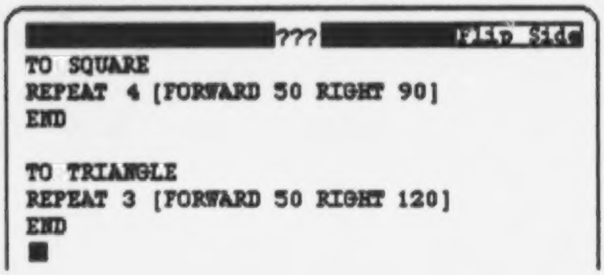


```
???                  Flip Side
TO SQUARE
REPEAT 4 [FORWARD 50 RIGHT 90]
END

TO SQUARE
REPEAT 4 [FORWARD 50 RIGHT 90]
END
■
```

Change the procedure name —>

Change the ———> numbers.

Finally, change the word SQUARE to TRIANGLE. You can do this easily by putting the cursor on the S of the word SQUARE, and pressing Erase to End of Line. (See Appendix 8.) Then type the word TRIANGLE. Next, change 4 to 3, and 90 to 120. You now have a TRIANGLE procedure with very little typing.



```
???                  Flip Side
TO SQUARE
REPEAT 4 [FORWARD 50 RIGHT 90]
END

TO TRIANGLE
REPEAT 3 [FORWARD 50 RIGHT 120]
END
■
```

At first, making copies like this may seem to take more time than typing them. However, with practice you will find that using Cut, Copy, and Paste saves you a lot of time.

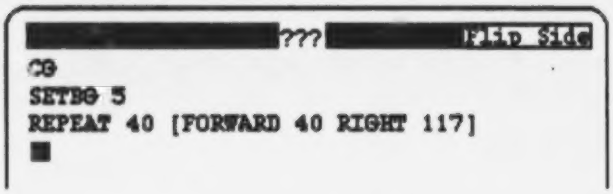
FROM THE COMMAND CENTER

These text moving commands work nearly anywhere in LogoWriter . Suppose that you typed the following commands in the Command Center:

```
CG
SETBG 5
REPEAT 40 [FORWARD 40 RIGHT 117]
```

and that you really like the effect created. You can use your new knowledge to make these commands into a procedure without retyping.

1. Place the cursor on the `C` of `CG`.
2. Press the Select keys.
3. Use the Arrow keys to highlight all three lines.
4. Press Copy. The highlighting disappears.
5. Press Flip to get to the back of the page.
6. Press Paste and the commands are ready to be made into a procedure.



```
???                  Flip Side
CG
SETBG 5
REPEAT 40 [FORWARD 40 RIGHT 117]
■
```

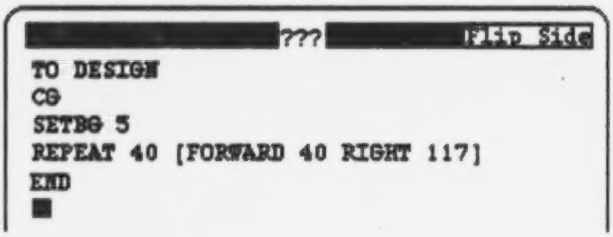
Add

```
TO DESIGN
```

and

```
END
```

and you have a procedure to create a design.



```
???                  Flip Side
TO DESIGN
CG
SETBG 5
REPEAT 40 [FORWARD 40 RIGHT 117]
END
■
```

Flip the page and type

```
DESIGN
```

and the design will appear on the page.

FROM ANYWHERE TO ANYWHERE

There are many places that the ability to move text can be useful. Here is a list of some ways that haven't been mentioned previously that you might want to use Select, Cut, Copy, and Paste. You may not be completely ready for this list yet, but you should return to it from time to time as you learn more about LogoWriter. In time, all of these ideas will become quite useful to you!

- Move a copy of a procedure from the flip side of a page to the front so that you can see the procedure and its effect at the same time. This can be useful if you are explaining how a procedure works to someone else.
- Move the commands from a procedure to the Command Center so that you can run them one at a time by putting the cursor on the first line and pressing Return/Enter. This can be *very* helpful in debugging.
- Move text written using the word processor into a procedure on the flip side of the page. If you are writing *about* a procedure, this method allows you to check yourself by actually running the lines.
- Move text around in the Command Center. You might want to move several sections of text together so that you can run each line without having to skip lines you no longer want.
- Move blocks of text from one page to another. You might want to move one procedure (or other text) from one page to another. Simply copy the text from the first page, go to the Contents page and get the page second page, then paste the text wherever you want it.

As you continue to work with LogoWriter, you will find many ways to use these word processing ideas to assist your Logo programming.

ON THE SHAPES PAGE

So far, you have used Select, Cut, Copy, and Paste with text. You can use Cut, Copy, and Paste on the Shapes page as well. First, try making a copy of a single shape from the Shapes page.

1. Go to the flip side of the Shapes page.
2. Use Next Screen or Previous Screen to go to a shape of your choice. (This action is the equivalent of using Select with text.)
3. Press Copy. Nothing seems to happen. Actually the shape you selected is not stored in a temporary place in the memory of the computer.
4. Use Next Screen or Previous Screen to go to a blank space/shape on the Shapes page.
5. Now press Paste. (You can also paste a "new" shape over one that is already there – but then the "old" shape is gone.)

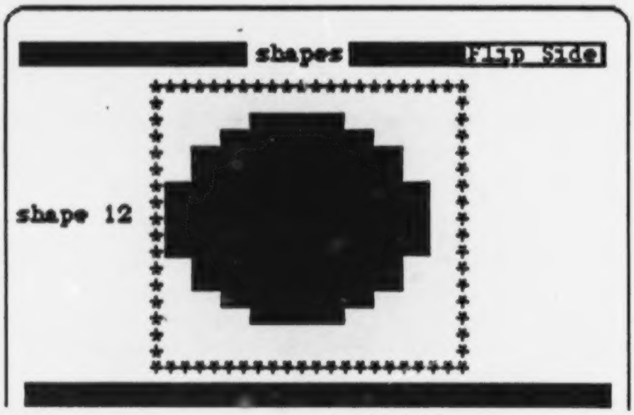
You see a copy of the shape you selected. Flip the page. You now have two copies of the shape you chose. Press Esc if you want to save this page.

You can also move a shape from one place to another.

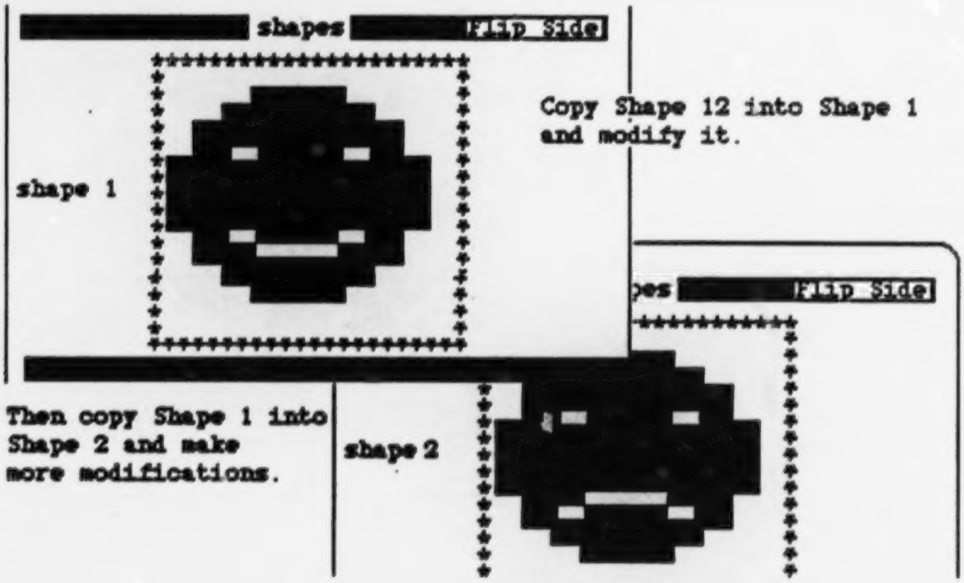
1. Go to the flip side of the Shapes page.
2. Use Next Screen or Previous Screen to go to a shape of your choice.
3. Press Cut. The shape disappears.
4. Use Next Screen or Previous Screen to go to a blank or unwanted space/shape on the Shapes page.
5. Now press Paste. The shape appears again.

Flip your page. You have now moved a shape from one spot to another. Press Esc if you want to save this shape.

Sometimes you want shapes that are very much alike. You can make one or more copies of a shape and then make small changes on the ones that need modification.



Copy this shape into shapes 1 and 2



Copy Shape 12 into Shape 1 and modify it.

Then copy Shape 1 into Shape 2 and make more modifications.

This avoids needing to copy a shape block by block, say from graph paper.

Using Cut , Copy, and Paste with shapes is powerful but potentially dangerous. Before you do much experimenting, be sure you have a shapes page to work with that doesn't have anything important on it. Then spend some time practicing. Only through practice will you find this process easy enough to be useful.

Keep in mind that when you press Esc, the Shapes page that you are working with is saved on the disk in the disk drive. It is easy to accidentally erase a Shapes page that you want this way. Think carefully before you press Esc on the Shapes page.

Sometimes you want to copy a shape from one Shapes page to another or from one disk to another. Copy is the "tool" that you use to accomplish this.

1. Put the disk containing the shape you want to copy in the drive and select Shapes.
2. Use Next or Previous screen to select the shape.
3. Press Copy.
4. Press Escape. (Be sure you do this *before* you switch disks or Shapes pages.)
5. Put the disk on which you want the shape in the disk drive.
6. Select the Shapes page.
7. Use Next or Previous Screen to move to the place where you want the shape.
8. Press Paste.
9. Press Escape to save the modified Shapes page.

You now have the shape on both disks.

TIPS AND TECHNIQUES

There are many ideas in this chapter that will be valuable to you in your future work with LogoWriter. It is important that you not only practice the ideas described here, but that you return to this chapter from time to time to review the suggestions that were made. Some of the ideas take a bit of practice to learn to use. For example, resist the temptation to continue retyping text rather than learning to us Copy and Paste. The effort put into learning these new techniques will pay off later.

When you are moving text or shapes, it is better to use Copy rather than Cut. If you make a mistake in Pasting the text or shape, you can always recopy it and try again. If you use Cut, then the shape or text is gone. After you have successfully Pasted, then Cut the text or shape you no longer want.

At the end of each line of text or the beginning of a blank line,there is an invisible "Return" or "End of Line" character. When you use Select, Cut, Copy or Paste, these characters will be moved about as well. Look at your screen carefully after you have selected text and you will see the highlighted character at the beginning of blank lines.

PROJECT IDEAS:

Use the word processor to write a couple of paragraphs for another class. Print a first draft. Then use the text-moving commands to modify it and print the final version.

Write a series of procedures to draw a triangle, square, pentagon, hexagon, etc. Use Copy and Paste so that you can do as little typing as possible.

From the Command Center, create an interesting design. Cut the commands from the Command Center to the back of the page and create a procedure (or procedures) from them.

Create a demonstration page. Write a procedure to make an interesting design. Then put a copy of the procedure on the front of the page along with an explanation that tells how the procedure produces the design.

Create a series of similar shapes. Perhaps it might be a man walking or a bird flying. Use Copy and Paste to begin each new shape.

Try reversing some shapes. For example, you might want an engine that runs to the left and another that runs to the right. A neat trick is to draw the shape on a transparency and then flip the transparency over. Can you use Copy and Paste to get your reversed shape started so that you don't have to enter every block?

15. Music

You have learned to work with text and graphics in LogoWriter. You can also make music! Try typing

```
TONE 440 50
```

You hear a musical note. Try changing the numbers after TONE. What does each one do?

Did you discover that the first number is the pitch of the note? The bigger the number, the higher the note. The second number is the length or duration of the note. The bigger the number, the longer the note.

The duration of a note is measured in twentieths of a second. That is,

```
TONE 440 20   plays for 1 second
TONE 440 60   plays for 3 seconds
TONE 440 10   plays for 1/2 second
```

To compute the length of a note, multiply the second input to TONE by 1/20.

The frequencies of the notes are the actual frequencies given in vibrations per second. The table at the bottom of the page gives the values for a number of octaves.

Using this chart, you would play a C scale (the white notes on the piano) for the octave starting with middle C using these commands:

```
TONE 262 10
TONE 294 10
TONE 330 10
TONE 349 10
TONE 392 10
TONE 440 10
TONE 494 10
TONE 523 10
```

There are no sharps in this scale. If you have music that uses flats, you need to know, for example, that A sharp is the same note as B flat. (Black notes on a piano.are sharps and flats.) You then use the chart accordingly.

Suppose you want to write a program to play a simple tune. You can write a procedure for each of the lines of the song. For example, to have LogoWriter play "Three Blind Mice," you can begin with

```
TO THREE.BLIND
TONE 330 10
TONE 294 10
TONE 262 20
END
```

This phrase is repeated twice, so you can write

```
TO LINE1
REPEAT 2 [THREE.BLIND]
END
```

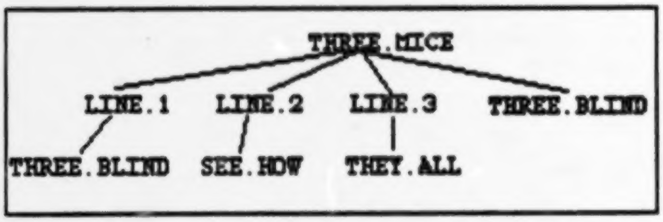
Similarly, the second line can be written as follows.

OCTAVES	NOTES										
	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
1			37	39	41	44	46	49	52	55	58	62
2	65	69	73	78	82	87	92	98	104	110	117	123
3	131	139	147	156	165	175	185	196	208	220	233	247
4	*262	277	294	311	330	349	370	392	415	440	466	494
5	523	554	587	622	659	698	740	784	830	881	932	988
6	1047	1109	1176	1244	1319	1398	1480	1566	1663	1761	1864	1973
7	2095	2213	2346	2495	2637	2797	2959	3142	3327	3510	3743	3946

*This is Middle C

```
TO LINE2
REPEAT 2 [SEE.HOW]
END

TO SEE.HOW
TONE 392 10
TONE 349  5
TONE 349  5
TONE 330 20
END
```

Can you finish this song? The procedure tree might look like this.


THREE.MICE
LINE.1
LINE.2
LINE.3
THREE.BLIND
THREE.BLIND
SEE.HOW
THEY.ALL

Using some of the words for the song helps you keep track of what procedures play what part of the song. In each of the "LINE" procedures above, a phrase of music is repeated, sometimes with the same words; sometimes with different words.

Even if you are not "musical" you can write music using LogoWriter. With some simple sheet music and a chart to translate the symbols on a musical staff into frequencies, you too can be a composer!

TIPS AND TECHNIQUES

The TONE command can be used for sound effects as well as music. Experiment with small and large numbers for the frequency to get buzzes

```
TONE 40 10
```

and squeaks

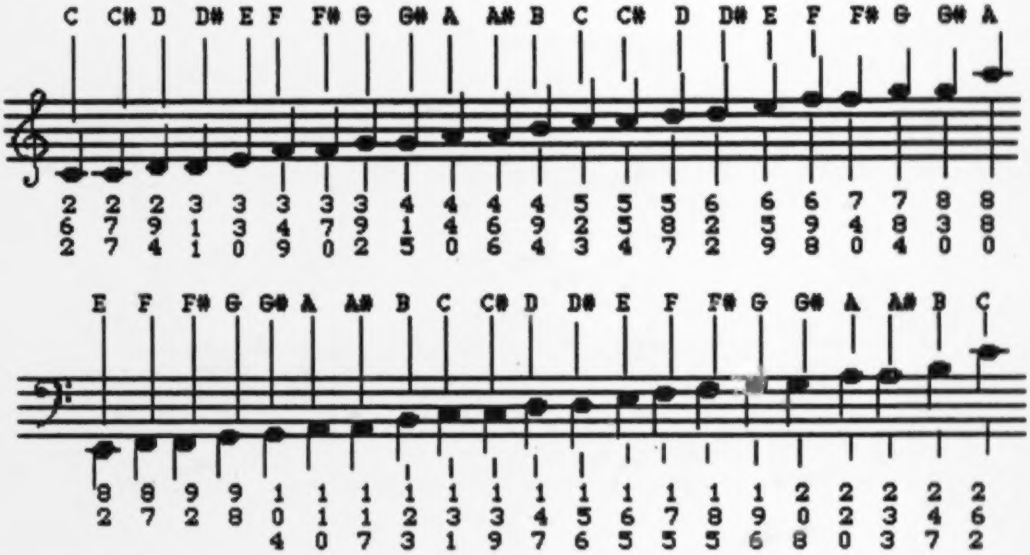
```
TONE 9000 20
```

and slowly changing tones

```
TONE 100 1
TONE 105 1
TONE 110 1
TONE 120 1
TONE 125 1
```

The sounds you can make using the TONE command can be quite varied.

Many people say "Oh, I'm not musical. I can't use Logo to make music." In fact, it is much easier to use a computer to make music than it is to play the piano or most instruments. To get started, find some beginning instrumental music. You might ask the music department in your school for some beginning music in the key of C. Match the notes on the music with the chart below.


C
C#
D
D#
E
F
F#
G
G#
A
A#
B
C
C#
D
D#
E
F
F#
G
G#
A
262
277
294
311
330
349
370
392
415
440
466
494
523
554
587
622
659
698
740
784
830
880
E
F
F#
G
G#
A
A#
B
C
C#
D
D#
E
F
F#
G
G#
A
A#
B
C
82
87
92
98
104
110
117
123
131
139
147
156
165
175
185
196
208
220
233
247
262

When you have identified the notes by name (A, B, C, etc.) use the chart in the first part of this chapter to write the frequencies next to each note. You will want to experiment with the length of a "quarter note", but the following chart shows you the **relationship** among notes. The length of a quarter note is quite arbitrary. However, once you choose the length for a quater note, then you divide by two to get an eighth note, double to get a half note, etc.

Note	Name	Duration
♪	Eighth note	8
♩	Quarter note	16
♩.	Dotted quarter note	24
𝅗𝅥	Half note	32
𝅗𝅥.	Dotted half note	48

Then you are ready to enter any song for which you have music.

PROJECT SUGGESTIONS

If you are involved with music outside of this class, pick a piece of music that you have sung or played and "teach" LogoWriter to play it!

Writing music in LogoWriter is an excellent way to practice writing modular programs. Select a piece of music and spend some time dividing either the written musical score or the words into small, meaningful sections. Each section will then be a procedure. Draw a procedure tree for your song. Then, when you write your song you will have an easy-to-debug program because you can check each part of the song as you go. (Note that this is using a strict top down approach to solving the problem of writing your song in LogoWriter.)

Try adding music to a page you have previously created. For example, you might have a birthday greeting that plays "Happy Birthday."

16. Multiple Turtles

By now you have done a lot of drawing with the turtle. You know that you can create designs, change colors, and stamp lots of different shapes using the turtle. But LogoWriter has more than one turtle. You have seen only turtle number 0.

When you start LogoWriter, you see the familiar turtle, turtle 0. Now type

```
TELL 1
ST
```

and a second turtle appears. Next type

```
TELL 2
ST
```

```
TELL 3
ST
```

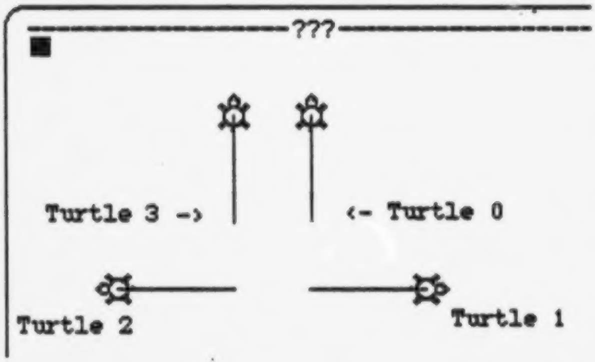
Now you see all four turtles. They form a square on the screen. The TELL command allows you to talk to whichever turtle you want. Now try this:

```
TELL 0
FORWARD 50
```

```
TELL 1
RIGHT 90
FORWARD 50
```

```
TELL 2
LEFT 90
FORWARD 50
```

```
TELL 3
FORWARD 50
```



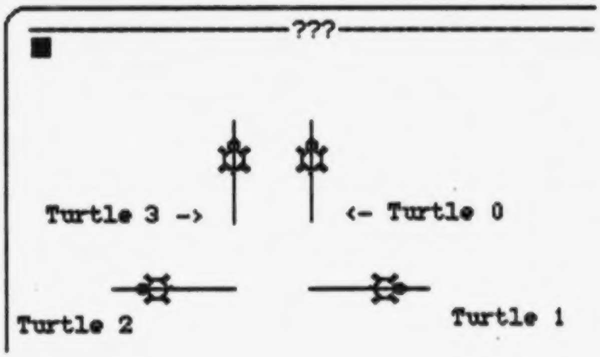
Notice how each turtle moves only when "spoken to" with the TELL command. Now type

```
TELL ALL
SETC 5
```

All of the turtles change color. Next type

```
BACK 20
```

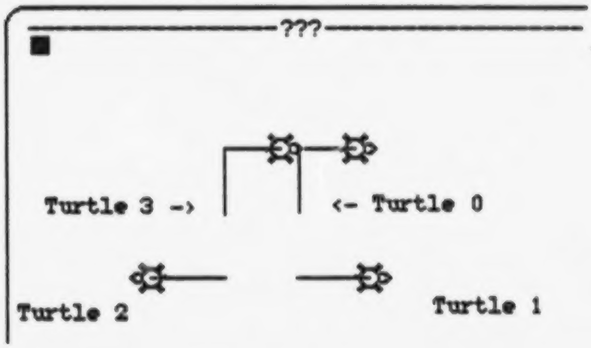
All of the turtles move backwards.



Now type

```
TELL [0 3]
RIGHT 90
FORWARD 20
```

Now two of the turtles move.



You can talk to any combination of turtles by putting their numbers inside square brackets. Be careful to use [and], not parentheses (and).

Now type

```
TELL ALL
HOME
```

HOME is a command that sends the turtles back to their starting places. Turtle 0 goes back to the center of the page. The others go back to their original places at the corner of a square.

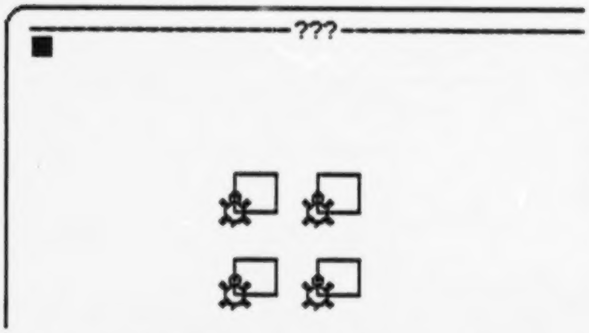
Try writing a simple procedure such as

```
TO SQUARE
REPEAT 4 [FORWARD 20 RIGHT 90]
END
```

Then type

```
CG
TELL ALL
ST
SQUARE
```

All four turtles draw a square.



It can be a lot of fun to have all of the turtles active at once! However, you should keep in mind as you are working with multiple turtles that it doesn't make sense to use several turtles just because they are available. For example, there is no reason to use two different turtles to stamp different shapes on the page. On the other hand, if you want two turtles *active* at the same time, then you need the capability of using more than one turtle. You will learn more reasons for having multiple turtles when you learn about animation (Chapter 20).

Sometimes when you are working with graphics, especially when you have many turtles, you want to get the page back to the way it started: a clear page, one turtle, black background, and white pen color. Remember the LogoWriter command to do this.

```
RG
```

that stands for Reset Graphics. Keep this in mind when you are working with multiple turtles.

TIPS AND TECHNIQUES

The most important concept to keep in mind when you begin working with multiple turtles is the meaning of the TELL command. TELL "wakes up" a turtle. It does nothing else. TELL does not make the turtle appear. You must type ST. It does not make the turtle draw. You must use FORWARD or BACK. It does not turn the turtle. You must use RIGHT or LEFT.

You also need to keep in mind that a TELL command remains in effect until you use another TELL command. If you use TELL [1 3] then turtles 1 and 3 will obey *all* turtle commands that you use until you "wake up" other turtles with a different TELL command.

Remember that you must use TELL 0 to get the "regular" turtle active again.

It is easy to become confused when working with multiple turtles. You can lose track of which turtle you are using. WHO gives you the numbers of the active turtles. If you type

```
SHOW WHO
```

then a list of active turtles appears in the Command Center. For example if you type

```
TELL [1 3]
```

and then type

```
SHOW WHO
```

you see

```
[1 3]
```

in the Command Center. You can put SHOW WHO in a procedure using multiple turtles to help in your debugging.

PROJECT SUGGESTION

There are many interesting things that you can do with multiple turtles. You can create fascinating page patterns using some or all of the turtles at once. Try writing a procedure to create an interesting graphics design. Then have all of the turtles draw it at once. You may want to write a SETUP procedure that places each of the turtles before they begin drawing.

17. Logo Grammar

Until now you have been writing Logo code at a rather intuitive level. You have been given no syntax rules as guidelines. Instead, you have followed the models given and experimented as needed, trying to understand error messages that appeared when you made a mistake. This is the way most people, both children and adults, begin their learning of Logo. Initially, it is quite easy to proceed this way. Logo was designed to provide an easy entry point, what Papert calls a "low threshold."

However, to have a more complete understanding of the Logo programming language, you need to understand its basic rules of grammar and punctuation. Once you understand these rules, you will find that learning new Logo primitives is easy and that you can pick up a technical reference manual for any version of Logo and understand how each primitive works. This level of understanding is necessary if you are going to progress beyond the beginning level in Logo programming. Take note that everything in this chapter applies to nearly any version of Logo that you encounter — it is in no way specific to LogoWriter.

On the other hand, you can't expect to completely understand these rules on first reading – or even the second reading! You should return to this chapter of the book from time to time. Gradually you will see how each primitive and concept fits into this larger picture.

LOGO OBJECTS

There are three kinds of "objects" that make up the Logo language: **words**, **numbers**, and **lists**. These three familiar terms have the following technical meanings in Logo:

word: an ordered collection of characters
number: a special kind of word that does not have to be quoted
list: an ordered collection of words and/or lists

The definition of "word" is not unlike its common meaning in English. However, a Logo word *can* contain symbols such as &, >, and +. In practice, however, you will most often see Logo words containing just letters, numbers, and periods. Usually a space separates two different Logo words. The name of any Logo procedure must be a Logo word. For example, SQUARE, or MOVE.TO.ROOF are both legal names of procedures. However, a procedure *cannot* have the name MY HOUSE because a space separates the title into two separate Logo words.

The definition for number contains yet another technical term. To **quote** a Logo object means to cause it to be evaluated to itself. That is, nothing changes when Logo encounters a quoted object. For example, if you type

```
PRINT "HI
```

you are telling Logo to print on the page the result of "HI. Since the quotation mark is the symbol for quoting a word, HI is evaluated to itself, and the word HI is printed on the page. If, however, you type

```
PRINT HI
```

There is no "HI" primitive, so unless you have defined a procedure named HI, Logo is unable to locate anything to **evaluate**. Thus, Logo responds with "**I don't know how to HI**" because it finds no procedure named HI.

This idea of quoting is prevalent both in English and in other programming languages. When you say "The word "cat" occurs 10 times in this paragraph" you are referring to the symbols c-a-t, not the soft, furry animal. If you type

```
PRINT "Hello there!"
```

in BASIC, or

```
WRITELN ('Hello there!');
```

in Pascal, you want the characters between the quotation marks printed exactly as they appear. In English and BASIC, we quote using " ", in Pascal, we use ' ', in Logo we use " before a Logo word and [ ] to surround a Logo list. More about these punctuation marks later.

Now, think again about the definition of a Logo number as being a special kind of word. In Logo you can type

```
PRINT 234
```

and the number 234 is printed, exactly as if you had typed

```
PRINT "234
```

The message "**I don't know how to 234**" does not appear. You don't have to quote numbers in Logo. This special case exists so that Logo can use standard arithmetic notation. That is, you can type

```
PRINT 5 + 7
```

instead of

```
PRINT "5 + "7.
```

Finally consider the third term in this chapter: list. You may have noticed something strange about this definition. It breaks the rule that you were no doubt taught in school about not using the word you are defining in the definition of that word. The definition of a Logo list is a special kind of definition. It is a **recursive definition**; a definition in terms of itself. You will learn more about this idea of recursion later. To give you an intuitive feel for what lists look like, here are some examples of **quoted** lists:

`[cat dog rat hat]` <— a list of 4 words

`[23 j qwaklfsoifalko]` <— a list of 3 words

`[cat [dog] rat]` <— a list of 2 words and 1 list (which contains one word)

`[[cat dog] [rat mouse] [house hat top]]` <— a list of three lists

LOGO PUNCTUATION

In the above discussion, there were a a couple of punctuation marks used. Let's formalize their meaning plus the meanings of a couple of others that are frequently used in Logo:

" <— used to quote a word

[ ] <— used to quote a list

( ) <— used as grouping symbols;
e.g., (PRINT "HELLO "GOODBYE)
or PRINT (93 + 75 + 86) / 3

: <— used to designate the value associated with a name
(This symbol will be discussed in Chapter 26 of this booklet.)

Too often when people are learning Logo, they ask questions such as "Should I use brackets or parentheses here?" Perhaps you have asked those questions as you were debugging your programs. Initially, that is OK, but as understanding grows, the above question should be rephrased as "Do I want a quoted list or do I want to group some part of an instruction here?" As you are asking yourself questions when you debug future programs, try rephrasing your questions in terms such as these. Return to this chart to remind yourself what each of these symbols means. Try to refrain from guessing what symbol you should use when you get an error message.

CATEGORIES OF PROCEDURES

All procedures in Logo fall into only two categories. You have already learned that you can describe a procedure as either a primitive (something that Logo "knows") or a user-defined procedure (a procedure you write). Both primitives and user-defined procedures can be subdivided into two groups: commands and reporters (another word for reporter is operation).

A **command** is a procedure that causes an effect.

A **reporter** is a procedure that "returns" a value to Logo.

The procedures that you have written to date are all commands, but you have used at least one reporter and will be learning more soon. Consider these examples.

If you type

```
FORWARD 50
```

the turtle moves forward 50 turtle steps. FORWARD is a command; its effect is to move the turtle the number of steps given as input to FORWARD. However, if you type

```
RANDOM 50
```

Logo responds with **"I don't know what to do with 37"** (or some other number less than 50). RANDOM is a **reporter**. It reports the result of evaluating RANDOM 50 "back" to Logo. You must tell Logo what to do with this result. For example, you might type

```
FORWARD RANDOM 50
```

At this point it is important to recognize that every Logo instruction must begin with a command. There can be any number of reporters after that command. However, if the first thing in a Logo instruction is a reporter, Logo returns the "**I don't know what to do with...**" message. If you get that message when you are programming, be sure that the instruction line you are working with begins with a command.

There is a simple way to understand the workings of any procedure, be it a primitive or user-defined. You need only answer these four questions:

1. Is it a command or a reporter?
2. How many inputs does it need?
3. What type or kind of Logo object (word, number, or list) must each of the inputs be?
4. If the procedure is a command, what is its effect?
   If the procedure is a reporter, what is its output—what does it report?

This series of questions comes from Chapter 2 of Brian Harvey's book, *Computer Science Logo Style, Volume 1*, (MIT Press), if you would like to examine a more thorough discussion of Logo grammar.

Practice these questions on some primitives or procedures that you have written. For example, PRINT.

1. It is a command.
2. It takes one input, unless it is surrounded by the grouping symbols () .
3. It accepts any Logo object: word, number or list.
4. Its effect is to place its input on the page at the current position of the cursor and then move the cursor to the next line.

Next, look at RANDOM.

1. It is a reporter.
2. It takes one input.
3. It requires a number as input.
4. It reports a number from 0 to one less than the number input.

Can you answer these questions for other Logo primitives — or some of your own procedures?

SUMMARY

If you are a bit overwhelmed, don't worry. These ideas will become clearer as you do more work with Logo. In summary, everything in Logo is a **word, list, or number.** Every procedure, be it user defined or a primitive is either a **command** or a **reporter.**

There are four kinds of punctuation marks used in Logo:

" is used to quote a word
[ ] are used to quote a list
( ) are used to group parts of an instruction
: is used to refer to the value of a name

TIPS AND TECHNIQUES

The material in this chapter is *central* to your understanding of Logo. It is unlikely that you fully understand the material in this chapter now. Understanding will come, however. Keep in mind that you should use the material in this chapter as reference, particularly as you encounter new primitives that you have difficulty understanding.

PROJECT IDEA

Select several Logo primitives that you feel comfortable using. Can you answer each of the four questions given at the end of this chapter? This is a good activity to do with a friend. Perhaps you could also practice describing each other's user defined procedures. You can check each other and discuss your disagreements. This kind of activity will help you understand Logo grammar much better.

18. Using READCHAR

You have, no doubt, used a computer program that asked you a question and then responded depending on what you typed. Perhaps it asked for your name and then greeted you by name. Perhaps you chose a number from a menu. Programs that you "talk to" are called **interactive programs**. You can write interactive programs in LogoWriter.

Recall that LogoWriter has four turtles, numbered 0, 1, 2, and 3. You can write a program that allows the user of the program to tell which turtle s/he wants to use. In order to do this, you need to use READCHAR. READCHAR is a reporter. It reports to Logo whatever is typed on the keyboard. Try typing

```
READCHAR
```

Nothing happens. If you look carefully, you see that the cursor in the Command Center disappears. Next press the number 9. LogoWriter responds with

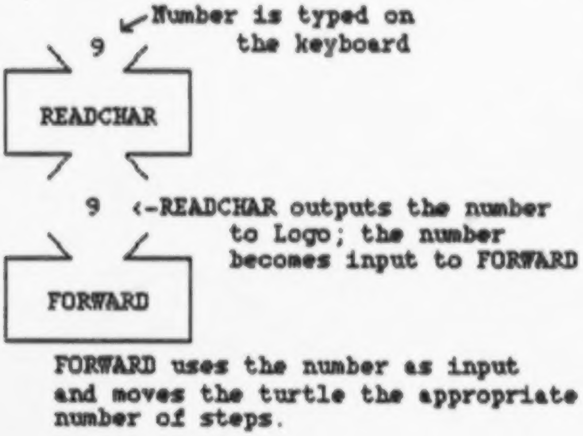
```
I don't know what to do with 9
```

In the last Chapter you learned that each Logo instruction needs to begin with a command. READCHAR is a reporter. When you type READCHAR all by itself on a line, Logo first waits for a character to be typed, and then informs you that it doesn't know what to do now.

Next try typing:

```
FORWARD READCHAR
```

The cursor disappears as before. Again, type the number 9.



9
Number is typed on the keyboard
READCHAR
9 <-READCHAR outputs the number to Logo; the number becomes input to FORWARD
FORWARD
FORWARD uses the number as input and moves the turtle the appropriate number of steps.

The turtle moves forward 9 steps. READCHAR reported the number 9 to FORWARD. FORWARD then used 9 as input. Logo responded by moving the turtle 9 turtle steps. (Using the ideas from the last chapter, recall that, FORWARD is a command and the effect caused by FORWARD is to move the turtle.)

Take a few minutes to reread the previous discussion. It refers to many of the new terms given in the last chapter. Perhaps you might even want to go back to that chapter and read the definitions of reporter and command again.

If you know another computer language such as BASIC or Pascal, you are more likely to find interactive reporters confusing. In both BASIC and Pascal, you have a variable into which you put any value typed on the keyboard. This value is then available by using the variable. (More about such ideas in LogoWriter in Chapters 26 and 35.) READCHAR is *not* a variable. It does *not* hold the value for later use. Instead, it immediately reports the value to LogoWriter. The action of READCHAR can be summarized as follows:

1. The cursor in the Command Center disappears.
2. LogoWriter waits until a key on the keyboard is pressed.
3. The *first* key pressed is reported by READCHAR *immediately.*
4. If READCHAR has a command in front of it, that command tries to use the character reported. If there is no command in front of READCHAR, it reports to LogoWriter; LogoWriter responds with **"I don't know what to do with —"**

Next look at a use of READCHAR in a procedure

```
TO SEE.TURTLE
TELL READCHAR
ST
END
```

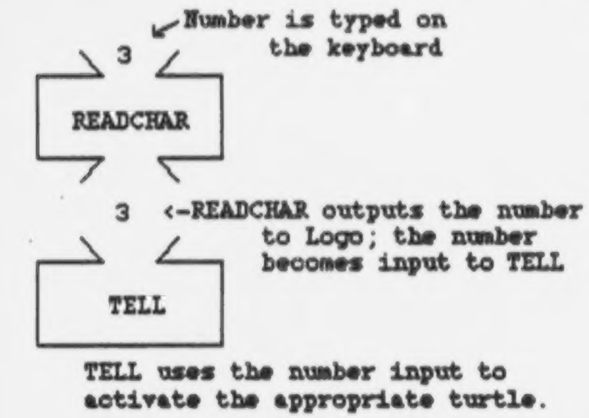
Type

```
SEE.TURTLE
```

The cursor disappears. Then type 1, 2, or 3. One of the turtles appears. Type

```
HT
SEE.TURTLE
```

and try a different number.


Number is typed on the keyboard
3
READCHAR
3 <-READCHAR outputs the number to Logo; the number becomes input to TELL
TELL
TELL uses the number input to activate the appropriate turtle.

Do you see that you can "talk to" any turtle you want using this procedure? In this case, READCHAR is reporting the number typed to the command TELL. TELL then uses the number to "activate" the appropriate turtle. When LogoWriter encounters the ST command, the active turtle appears.

You could also use READCHAR to change the color of the turtle's pen. For example,

```
TO PICK.COLOR
PRINT [WHAT COLOR DO YOU WANT?]
SETC READCHAR
END
```

or to pick a turtle shape

```
TO PICK.SHAPE
PRINT [WHAT SHAPE DO YOU WANT?]
SETSH 10 + READCHAR
END
```

Notice that in the last example, the only shapes that can be selected are numbers 10 through 19. READCHAR can accept only 1 digit: 0 through 9, so adding 10 gives you 10 through 19.

Here's a more elaborate example.

```
TO MAKE.SQS
SET.UP
WHICH.TURTLE
   SQUARE
END

TO SET.UP
CG
TELL ALL
HT
END

TO WHICH.TURTLE
PRINT [Which turtle do you want?]
TELL READCHAR
ST
END

TO SQUARE
REPEAT 4 [FORWARD 50 RT 90]
END
```

When you type

```
MAKE.SQS
```

the message

```
Which turtle do you want?
```

appears and you can then type the number of the turtle that you want to draw a square. As before, READCHAR reports to TELL. TELL activates the turtle. (The effect of TELL is to activate a turtle.) Then the turtle appears. Finally the active turtle follows the directions in the procedure SQUARE.

Notice the SET.UP procedure in this program. It makes sure that all of the turtles are hidden before you begin. It is a good idea to include a procedure to make sure the page is the way you want it before you start a program. Sometimes that means only a RG, sometimes CT and CG, and sometimes you have to worry about a number of turtles.

Later you will learn other ways to write interactive programs and other ways to use READCHAR. You should take some time now to be sure that you see how READCHAR works. While you are practicing, verbalize the ideas about commands and reporters that are used in this chapter. As you do so, your understanding of Logo grammar will increase.

TIPS AND TECHNIQUES

Keep in mind that you will *never* have READCHAR on a line by itself. Every Logo instruction must begin with a command and READCHAR is a reporter. READCHAR must report a value to a command.

You can only read one character at a time with READCHAR. Don't try to read numbers with more than one digit or words using READCHAR. Later you will learn reporters to read words or lists.

READCHAR does not save the value you type. Think of READCHAR as a "funnel" between the keyboard and a Logo command. If you use a funnel to pour liquid from one container to another, there is no liquid left in the funnel when you are done. If you use READCHAR to move a character from the keyboard to Logo, there are no characters "left in READCHAR."

PROJECT SUGGESTIONS

Write a multiple turtle program of your own that allows the user to tell which turtle s/he wants to use. You can have the turtle make elaborate or colorful designs.

Write a program that asks the user what color background s/he wants to use before drawing a design.

Write a program that asks the user what color the pen should be before drawing a picture of some kind.

Write a program that allows the user to set the turtle to any of the shapes from 0 to 9. (Later you will learn how to enter 2 digit numbers.)

Select any procedure that you have used that includes READCHAR. See if you can draw a diagram like the ones in this chapter to picture the action taking place in your procedure.

19. Indefinite Repetition

Whenever you wanted to repeat some action in LogoWriter, you used the REPEAT command. REPEAT is followed by the exact number of times you want the action to happen. So, if you want to draw a figure with 6 sides, a six is placed after the REPEAT. You must know exactly how many times you want an action to occur when you use repeat. This is sometimes called a definite loop.

Looping is a powerful idea in computing. Computers can do repetitive tasks endlessly without tiring. However, sometimes you do not know exactly how often you want an action to occur. For example, you might not know exactly how many people turned in ballots to be tallied. Or, you might not know how many people got overtime this pay period. You might have a design that you want to appear on the page until the user stopped the program. There are many examples in real life of indefinite loops.In this chapter, you will take the first step towards learning to construct loops in Logo that can be used to repeat some action an unknown number of times. To begin, you will learn to create loops that go on forever

Suppose you have the following procedure.

```
TO DRAW
SETC RANDOM 6
FD 50
RT 177
END
```

If you type

```
DRAW
```

and then move the cursor up and run DRAW again a number of times, a "fluffy ball" design begins to form.

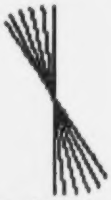
You could then write the procedure

```
TO DESIGN
REPEAT 50 [DRAW]
END
```

to create the "fluffy ball." You need to do some experimenting until you get just the right number after REPEAT to produce the design that you wanted.

An interesting effect occurs if you repeat the procedure DRAW over and over an over as different colors change randomly. It is quite easy to have this procedure repeat an indefinite number of times. You can write

```
TO DRAW
SETC RANDOM 6
FD 50
RT 177
DRAW
END
```



Now when you type DRAW, the effect goes on and on. Do you see what is happening? Use Stop keys (See Appendix 8) to stop procedure DRAW.

1. Procedure DRAW is called.
2. A random pen color is picked.
3. The turtle goes forward 50 "turtle steps."
4. The turtle turns right 177 degrees.
5. Procedure DRAW is called...but this is the same as step (1)!

Thus procedure DRAW calls procedure DRAW indefinitely! Procedure DRAW contains a call to itself; it is said to be **recursive.** The command DRAW just before the word END is referred to as the **recursive call** to procedure DRAW. This is an example of the simplest kind of recursion, called **tail recursion.** You will learn more about recursion later in this book (Chapter 21).

Here's another idea to try. Suppose that you want to leave a dramatic message on the computer screen as a reminder to a friend. You could write a recursive procedure to accomplish this task.

```
TO REMIND
PRINT [Don't forget the dance on Saturday!]
PRINT []
WAIT 10
REMIND
END
```

Now, if you type REMIND, the message will continue to appear over and over until someone presses the Stop keys.

Infinite loops are fun to use to get flashy effects — but they are not particularly useful in and of themselves. The fact that a procedure can call itself is a powerful idea. Not all computer languages allow recursion. (For example, BASIC.) You will learn how to use recursion to build more practical loops later on in this book.

One word of "warning" is in order now. For now, when you write recursive procedures, be sure that recursive calls are always the last line before END. Otherwise, Logo is never able to get to the end of your procedure and you may get some very unexpected results. For example

```
TO OOPS.DRAW
SETC RANDOM 6
FD 50
OOPS.DRAW
RT 177
END
```

If you run this procedure, you will no longer get a "fluffy ball". This procedure functions as follows:

1. Procedure OOPS.DRAW is called.
2. A random pen color is picked.
3. The turtle goes forward 50 "turtle steps."
4. Procedure OOPS.DRAW is called.

Logo is never able to get to the RIGHT 177. The procedure does not run as you expect. This placement of OOPS.DRAW before the end of the procedure OOPS.DRAW is called **embedded recursion** and you will learn more about it later. In the meantime, keep in mind that your recursive procedure calls should always be just before the END of the procedure.

TIPS AND TECHNIQUES

The important idea in this chapter is *looping*. For now the infinite loops that you can write using recursion are not really useful. It is better not to try to use them in programs at this point. Instead, simply practice the idea so that you understand that a procedure can call itself.

Embedded recursion can cause many bugs in programs. Be very careful when you start using recursive calls in programs. Make sure the recursive call is on the last line of the procedure. Nothing should be after the recursive call if you are using tail recursion to build loops.

You can build loops that use several procedures. For example

```
TO DRAW.LINES
FORWARD 50
RIGHT 177
CHANGE.COLOR
END

TO  CHANGE.COLOR
SETC RANDOM 6
DRAW.LINES
END
```

DRAW.LINES is still a tail recursive procedure call. Most likely you would write the above as one procedure. However, if your procedures get very long or complex, you may want to subdivide them. Then the recursive call to the "first" procedure is on the last line of the "last" procedure.

PROJECT SUGGESTION

Create a procedure that has an interesting effect. You might display a flashy graphic design, play a line of music, or create some mixture of graphics or sounds. Turn your procedure into an indefinite loop. Next, make your procedure part of a larger program, but make sure that your looping procedure is the last procedure you use in your program. Otherwise, your program will never get beyond your procedure containing your indefinite loop!

20. Animation

No doubt you have seen computer programs that have objects moving across the screen. If you have played video games, you often see objects, characters, or space ships in constant movement. Using what you know about making shapes and about looping, you can write procedures to cause movement on the page.

Begin by starting LogoWriter and getting a new page. Type these commands:

```
PU
SETSH 25
FORWARD 2
```

Use the arrow keys to repeat the last command a number of times. The helicopter moves slowly to the top of the page. You can put this in a program that includes an indefinite loop.

```
TO LIFT.OFF
SET.UP
GO.UP
END

TO SET.UP
CG
PU
SETSH 25
END

TO GO.UP
FORWARD 2
WAIT 2  <—This statement keeps the helicopter from
GO.UP              moving too fast.
END
```

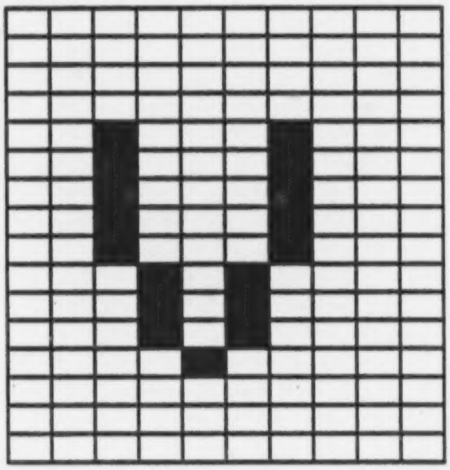
This program has some problems because the helicopter ends up wrapping around the bottom of the page. It looks like it is flying right through the ground! Perhaps a better approach would be to modify the SET.UP procedures as follows:

```
TO SET.UP
CG
PU
SETSH 25
RIGHT 90    <— This line is changed.
END
```

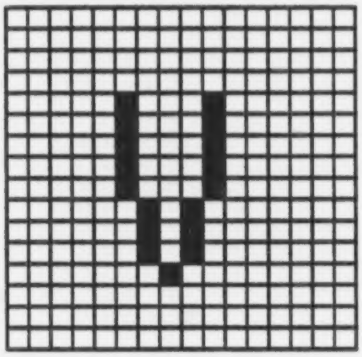
Now try the FLY procedure. The helicopter moves to the right. You can easily change the speed at which the helicopter moves. Make the number after WAIT larger. Make it smaller. (You can use decimals.) If you just use FORWARD by itself, you have no control over the speed. Take some time to experiment with these ideas. You should become comfortable changing the inputs to both WAIT and FORWARD to adjust the speed of animation.

The movement of the helicopter involves only one shape. Animation of the sort you see in cartoons and in many computer programs often involves changing of the shape as it moves. You can do this in LogoWriter by creating several shapes.

Go to the Shapes page and define Shape 1 as shown below.



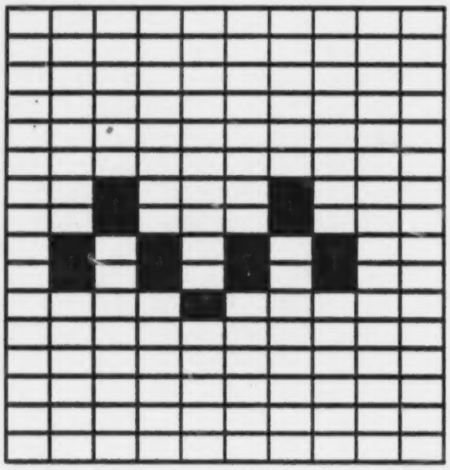
Shape 1 (Apple)



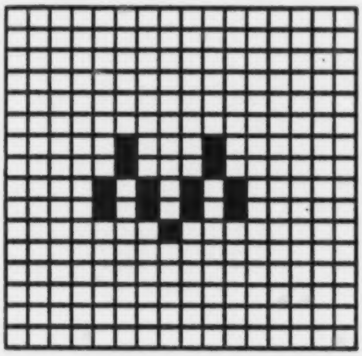
Shape 1 (IBM or Commodore)

Using the Copy keys, copy this shape. Move to Shape 2 with the Next Screen. Use the Paste keys to paste it in place. Modify Shape 2 so that it looks like this.

Shape 2 (Apple)

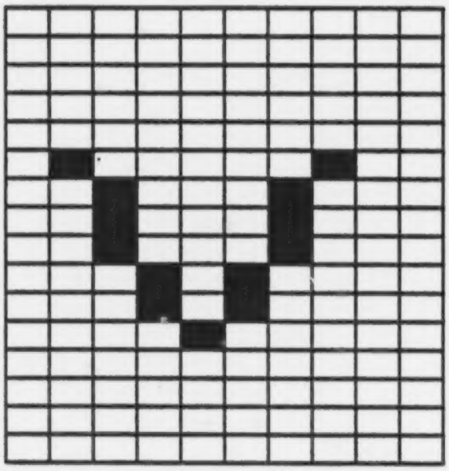


Shape 2 (IBM or Commodore)

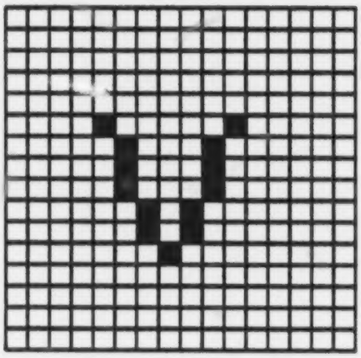


Repeat this process to do Shape 3 as shown in the next column.

Shape 3 (Apple)



Shape 3 (IBM or Commodore)



You can check the effect of this animation by using Next and Previous screen quickly in succession. Go to Shape 1. Press Next Screen twice quickly and then press Previous Screen twice quickly. Repeat this several times and you will see the effect of your animation without leaving the Shapes page.

Now press Esc and go to your page. Write this procedure on the flip side:

```
TO FLAP
SETSH 1
WAIT 2
SETSH 2
WAIT 2
SETSH 3
WAIT 2
SETSH 2
WAIT 2
SETSH 1
WAIT 2
FLY
END
```

Now type

```
FLAP
```

Do you want the bird to flap its wings faster or slower? If so you can change the numbers after the WAIT commands.

To make the bird move as well as flap its wings, add some FORWARD commands:

```
TO FLAP
SETSH 1
FORWARD 1
WAIT 2
SETSH 2
FORWARD 1
WAIT 2
SETSH 3
FORWARD 1
WAIT 2
SETSH 2
FORWARD 1
WAIT 2
SETSH 1
FORWARD 1
WAIT 2
FLY
END
```

Now type

```
PU
RIGHT 90
FLAP
```

To make the bird fly faster or slower, you can change the numbers after the WAIT or the FORWARD commands. Then you can really make the bird fly by writing this procedure.

```
TO FLY
REPEAT 20 [FLAP]
END
```

If you want the flying to continue indefinitely, then you can write this procedure.

```
TO FLY.FOREVER
FLAP
FLY.FOREVER
END
```

It is fun to combine music and animation, but it is more complex than it seems at first thought. Keep in mind that the computer can do only one thing at a time. To play music while animation occurs requires some planning and careful programming.

Begin by looking at a simple example. Suppose you would like to have the bird fly while the notes of the scale play. You can go back to the FLAP procedure and insert a note between each flap of the wings.

```
TO FLY.AND.SING
TONE 262 10
FLAP
TONE 294 10
FLAP
TONE 330 10
FLAP
TONE 349 10
FLAP
TONE 392 10
FLAP
TONE 440 10
FLAP
TONE 494 10
FLAP
TONE 523 10
FLAP
END
```

A similar technique must be used if you want to animate two different turtles. You must TELL one turtle to move a bit and then TELL the second turtle to move a bit. Then you repeat the sequence over and over, alternating between turtles.

Be careful when writing procedures such at this. It is easy to create very long, difficult-to-debug programs. Remember to break your procedures into small, meaningful parts. When you have learned more Logo programming, you will see that procedures like FLY.AND.SING can be written much more efficiently.

Using the methods described above, you can create a wide variety of interesting movement — as well as sound. Animation is a lot of fun and can add interest to any project. With the ability to create the shapes of your choice, your projects are limited only by your own creativity.

TIPS AND TECHNIQUES

By now you are writing more complex programs. As mentioned above, you should always keep your procedures short and give them meaningful names. This makes debugging easier.

Another helpful hint for debugging is to make sure each procedure works by itself. Don't always type the name of the top level procedure and run the entire program. Run *each* procedure and make sure that it works. Run the entire program only when you are sure that all of the parts work.

When you have a lot of procedures, it is easy to lose track of the action of each procedure. One way to keep track of which procedure is running is to put PRINT statements in each procedure. For example

```
TO FLY
PRINT [IN PROCEDURE FLY]
REPEAT 20 [FLAP]
END
```

prints the words "IN PROCEDURE FLY" each time the FLY procedure is called. This gives you a trace of the action of the program. Putting in PRINT statements will slow down the action of your program and clutter your screen. However, it will help you see what is happening as your program runs.

PROJECT SUGGESTIONS

Create a street scene with houses and trees and have a car driving down the street. Perhaps you could add some sound effects as well.

Draw a picture of an airport and have an airplane flying by, or perhaps a helicopter landing...or both! (Hint: Use REPEAT to land the helicopter and then recursion to have the airplane continue to fly overhead.)

Define a number of shapes and create your own animated movie. Be careful to plan your program carefully so that none of your procedures get too long! How about adding a sound track to your movie?

21. Creating Definite Loops

So far when you have written procedures with recursive calls to repeat an action, you needed to use the Stop keys to interrupt the program. This is not very useful if you want the program to stop automatically or you want to have the user ask it to stop. To stop an indefinite loop, you need to learn about the IF statement. IF is a Logo primitive that allows you to make decisions based on some condition.

Begin by typing

```
SETBG 3
```

and then typing

```
SHOW BG
```

The number 3 appears in the Command Center. SHOW is a command used to display text in the Command Center. BG is a reporter that returns the number of the current background color. Now define this procedure:

```
TO CHANGE
SETBG RANDOM 6
SHOW BG
CHANGE
END
```

Type

```
CHANGE
```

and watch the background color change as the current background color numbers appear in the Command Center. Notice that SHOW can be used like PRINT to help debug procedures. (See Tips and Techniques in the previous chapter.)

Suppose you wanted to change the background color randomly until the background became black. Modify CHANGE as follows:

```
TO CHANGE
SETBG RANDOM 6
IF (BG > 0) [CHANGE]
END
```

The condition after IF is "BG > 0." You can then read the statement "if the background color is greater than zero, call the CHANGE procedure."

Examine the IF instruction more carefully. Notice that it is made of three parts:

1. The word IF.
2. A condition, in this case BG > 0.
3. A list of commands, in this case just CHANGE.

Logo checks to see if the condition (BG > 0) is true. If it is true, it runs the list of commands (CHANGE).

When you run CHANGE, you see that the background color changes continuously until it turns black. Remember that 0 is the color number for black. Sometimes the background will change color many times; sometimes not at all. The number of changes depends on the number reported by RANDOM.

Here's another way to stop recursion. Suppose you want to ask the user a question, such as "Do you want to see that design again?" The following procedure will do just that:

```
TO ASK.Q
DESIGN   <— This a procedure that you have defined elsewhere.
PRINT [Do you want to see another design?]
IF (READCHAR = "Y) [ASK.Q]
END
```

The three parts of this IF statement are:

1. The word IF
2. The condition: READCHAR = "Y
3. The action to take if the condition is true: call ASK.Q

Thus, this procedure

- Runs a procedure called DESIGN which is defined on the flip side of the page.
- Prints a message on the page: "Do you want to see another design?"
- Checks to see if the character reported by READCHAR is a Y.
- If the character is a Y, the procedure ASK.Q is called. If the character is not a Y, Logo goes to the END and stops.

Notice that the character Y is preceded by a quotation mark ("). Without the quotation mark, Logo would look for a procedure called Y. READCHAR reports a word made up of one character. You must quote the Y for Logo to compare what READCHAR reports with the letter Y. (This might be a good time to review the ideas discussed in Chapter 17.)

You will learn more about using IF later. For now, you can begin to use it to stop simple recursive procedures.

TIPS AND TECHNIQUES

When the condition after an IF statement is true, the list in the IF statement is run. When the condition is false, the *next* statement in the procedure is run. Later you will learn how to have two different actions happen *in* the IF statement: one when the condition is true; another when it is false.

The punctuation in IF statements can cause some trouble. Keep in mind that there are three parts to the IF statement. The word IF has no punctuation. The condition is the second part and may contain some punctuation such as parentheses. The third part is the list of actions to run. This list must be surrounded by square brackets. This can be confusing if you have a PRINT statement after the IF. For example

```
IF (BG = 0)
   [PRINT [THE BACKGROUND IS BLACK.]]
```

The statement

```
PRINT [THE BACKGROUND IS BLACK.]
```

has brackets around it when used in the IF statement. That means there are two square brackets at the end of the IF statement.

The condition in an IF must return true or false. One way to see if your condition is functioning correctly is to copy it into the Command Center and run it by itself. For example, if you put

```
BG > 0
```

in the Command Center, then Logo responds

```
I don't know what to do with true
```

or

```
I don't know what to do with false
```

If the message on your screen is anything else, then you have written your condition incorrectly.

There is no "not equals" sign in Logo. If you wanted to say "If the background is not equal to 0, call the procedure CHANGE" you would write

```
IF NOT (BG = 0) [CHANGE]
```

Simply put the word NOT in front the = sign.

When using standard math symbols such as >, < and = in conditions, it is a good idea to use parentheses. This has to do with the way Logo "parses" or translates the words you type so the computer can understand them.

Inserting PRINT or SHOW into procedures using IF statements can be very helpful in debugging. For example,

```
TO CHANGE
SETBG RANDOM 6
SHOW BG
IF (BG > 0) [CHANGE]
END
```

causes the number of the background color to appear in the Command Center before it is tested in the IF statement. Because this happens so fast, you might also want to insert a WAIT command.

```
TO CHANGE
SETBG RANDOM 6
SHOW BG
WAIT 5
IF (BG > 0) [CHANGE]
END
```

Be sure to put the PRINT or SHOW commands *before* the recursive call.

PROJECT SUGGESTIONS

Write a program that randomly changes the pen color and draws a design in that color until black is picked.

Write a program that displays a message on the page over and over until the user types something other than Y.

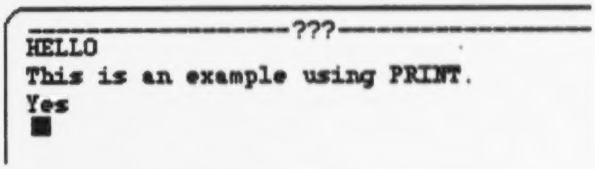
Write a program that randomly picks the shape of a turtle and stamps it on the page until the turtle shape (0) is picked.

22. Printing Text on the Screen

You have seen how to put text on the page using the PRINT command. You have also seen that SHOW can be used to put text in the Command Center. PRINT and SHOW are two of the four ways to put text on the screen in LogoWriter. You will now see how each of the four commands works.

First, recall that PRINT must be followed by something to print. Whatever follows PRINT must be quoted. That is, it must have a quotation mark in front of it if it is a word, or be enclosed in square brackets if it is a list. The input to PRINT then appears on the page. Here are some examples of PRINT:

```
PRINT "HELLO

PRINT [This is an example using PRINT.]

PRINT "Yes
```



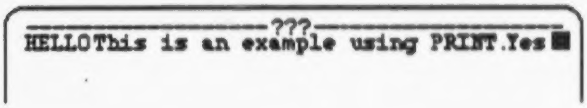
```
HELLO
This is an example using PRINT.
Yes
■
```

Note also, that when you use PRINT, the cursor moves to the next line after the input is printed.

There is another command that allows you to put text on the page. This command is INSERT. It works exactly like PRINT except that the cursor does not move to the next line after the input is printed. Here are some examples of INSERT.

```
INSERT "HELLO

INSERT [This is an example using PRINT.]

INSERT "Yes
```

If you type these commands, one after the other, the three inputs shown above will appear one after the other:



```
HELLOThis is an example using PRINT.Yes■
```

Notice that the cursor remains on the same line when you use INSERT.

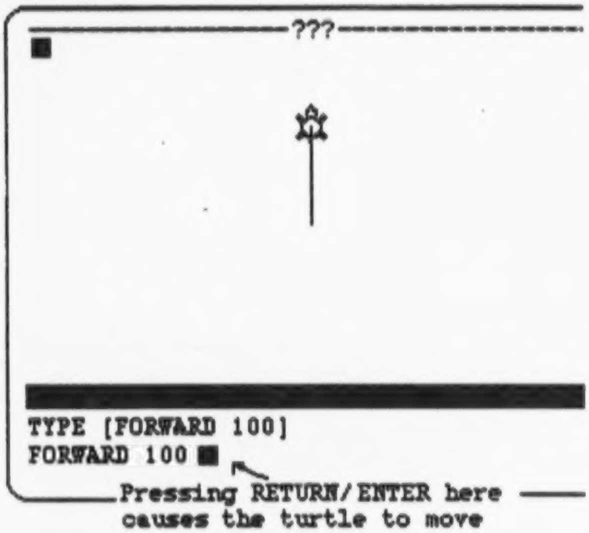
The other two commands for printing text are TYPE and SHOW. Both of these put text in the Command Center. TYPE works much like INSERT. That is, the cursor remains on the same line after the command is run. This can have unexpected results. Suppose you are in the Command Center and you type

```
TYPE [FORWARD 100]
```

Then

```
FORWARD 100
```

appears. Then, if you press Return/Enter, the turtle moves forward 100 steps.



```
TYPE [FORWARD 100]
FORWARD 100 ■
```

Pressing RETURN/ENTER here causes the turtle to move

You need to be especially careful when using TYPE.

You have seen SHOW briefly before. SHOW will be come particularly helpful as you work towards understanding Logo grammar better. SHOW prints its input in the Command Center including any square brackets. This allows you to tell if the input to SHOW is a Logo word or a Logo list. For example

```
SHOW "HELLO
```

produces

```
HELLO
```

while

```
SHOW [THIS IS AN EXAMPLE]
```

produces

```
[THIS IS AN EXAMPLE]
```

Being able to tell whether you have a word or a list is helpful as you begin to do more work with Logo lists. However, if you simply want to put a question in the Command Center, you don't want to use the square brackets. A clever combination of TYPE and SHOW can solve your problem:

```
TO COMMAND.CENTER.Q
TYPE [Here is my question]
SHOW "
END
```

TYPE presents the question without brackets; SHOW followed by a quotation mark prints an empty word. Another clever solution uses a Logo primitive you haven't seen before.

```
TO DIFFERENT.WAY
TYPE [Here is my question]
TYPE CHAR 13
END
```

Every character on your keyboard has an associated number that is used by the computer to represent that character. This number is called its ASCII value. The number 13 is the ASCII value of the Return/Enter key. CHAR takes an ASCII value as input and reports the corresponding character. Thus CHAR 13 reports the Return/Enter character and TYPE prints it, moving the cursor to the next line.

Notice that all four of these commands need one input. However, there are times when you want them to have more than one input. In the above example, you can eliminate the second TYPE statement by surrounding TYPE with parentheses.

```
(TYPE [Here is my question] CHAR 13)
```

The parenthesis tell Logo that there is more than one input to TYPE. Below is another example.

```
TO ASK.INITIAL
PRINT [What is your first initial?]
(PRINT READCHAR [is a nice initial.])
END
```

If you type

```
ASK.INITIAL
```

and then press S, Logo will print

```
S is a nice initial.
```

In this example, PRINT has two inputs: the character that READCHAR reports *and* the list [is a nice initial.] When you have more than one input for these commands, be sure the first parenthesis goes *before* the command and the last parenthesis goes after the last input.

TIPS AND TECHNIQUES

If you are writing interactive programs, using TYPE and SHOW to display questions for the user provides an excellent way to keep your page from being cluttered with text.

If you are debugging a program that uses PRINT as part of the program, try using SHOW to help with the debugging. Suppose you have

```
TO DRAW
DESIGN
PRINT [DO YOU WANT TO SEE THE DESIGN AGAIN?]
IF (READCHAR = "Y) [DRAW]
END
```

You might insert

```
TO DRAW
DESIGN
PRINT [DO YOU WANT TO SEE THE DESIGN AGAIN?]
SHOW [JUST BEFORE IF IN DRAW]
IF (READCHAR = "Y) [DRAW]
END
```

so that the diagnostic messages appear in the Command Center and don't get mixed up with the output from the program. Similarly, you could use PRINT to diagnose problems in a program using SHOW for output.

PROJECT SUGGESTION

Take some time to experiment with each of these commands. Make sure you understand how each works so that you will know which one to use if you are writing a program of your own.

23. More on Interactive Programming

You have learned how to use READCHAR to accept a single character from the keyboard. With it, the user can interact with the program. For example

```
TO PICK.COLOR
PRINT [What background color do you want?]
SETBG READCHAR
END
```

allows the program user to select the background color.

LogoWriter has two other primitives for writing interactive programs: READLIST and READLISTCC. Both are reporters. Both accept a list from the keyboard. Recall that a Logo list is a collection or words and/or lists. Anything you type can be accepted using either of these two new reporters.

USING READLISTCC

READLISTCC reads a list from the Command Center. Here is an example

```
TO GREET
CT
PRINT [What is your name?]
(PRINT [Glad to meet you, ] READLISTCC)
PRINT []
PRINT [What is your favorite color?]
(PRINT [I like ] READLISTCC [too!])
END
```

If you type

```
GREET
```

then

```
What is your name?
```

appears at the top of the page. If you type

```
George
```

Notice that George appears in the Command Center. Then, on the page you see

```
Glad to meet you, George
What is your favorite color?
```

In the Command Center, suppose you type

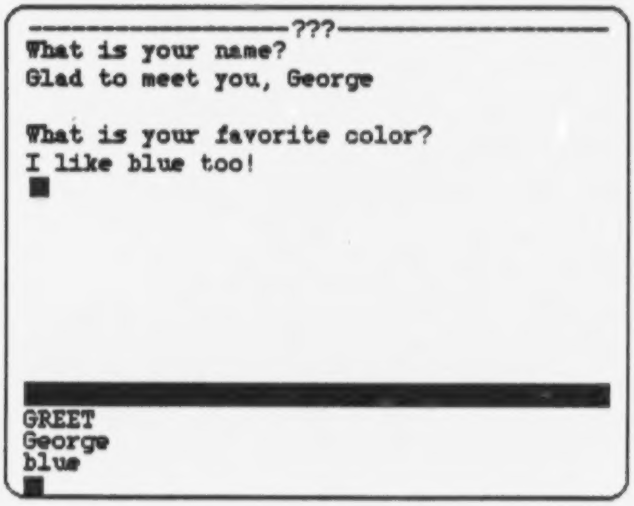
```
blue
```

then

```
I like blue too!
```

appears on the page.

Look carefully at the illustration at the bottom of this page.



The procedure produces these sentences.

The user of the program types these lines.

```
-----------------???-----------------
What is your name?
Glad to meet you, George

What is your favorite color?
I like blue too!
```

```
GREET
George
blue
```

Using READLISTCC allows you to accept responses that are not typed on the page.

USING READLIST

READLIST works differently from READLISTCC in that it accepts words and lists from the page rather than the Command Center. The following example is like the first one using READLISTCC. Take some time to be sure you see the difference:

```
TO GREET
CT
INSERT [What is your name?]
(PRINT [Glad to meet you, ] READLIST)
PRINT []
INSERT [What is your favorite color?]
(PRINT [I like ] READLIST [too!])
END
```

When you type

```
GREET
```

the page clears and

```
What is your name?
```

appears, with the cursor on the same line. Remember that INSERT leaves the cursor on the same line on the page. Suppose you type

```
George
```

On the page, you then see

```
What is your name? George
Glad to meet you, George

What is your favorite color?
```

The cursor stays on the same line. Suppose you type

```
blue
```

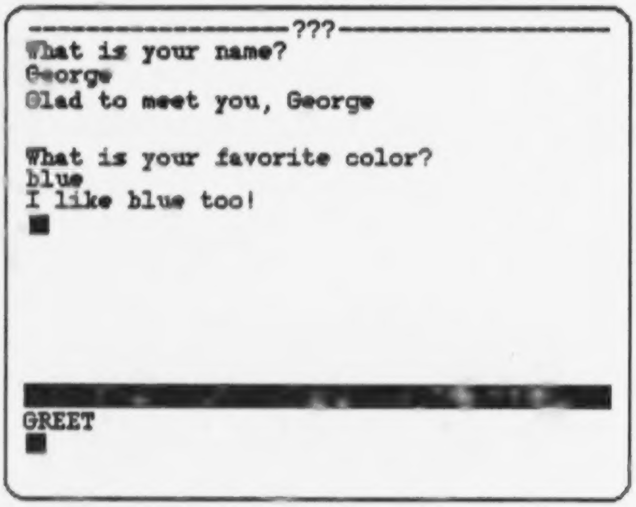
Then you see

```
What is your favorite color? blue
I like blue too.
```

Look carefully at the illustration at the bottom of this page. Compare this output with that produced by GREET using READLISTCC. Do you see that what is typed appears on the page twice when you use READLIST, but once on the page and once in The Command Center when you use READLISTCC?

CREATING READWORD PROCEDURES

You have learned to accept single character words from the keyboard using READCHAR. Those single character words could be numbers. (Recall that a number is a special kind of word.) You have also learned to accept lists from the keyboard using READLIST and READLISTCC. However, sometimes you want to use a single word or a number that has more than one character.



User types -->

User types -->

```
------------------???------------------
What is your name?
George
Glad to meet you, George

What is your favorite color?
blue
I like blue too!
█

GREET
█
```

If you want to work with words of more than one character, you can't use either READCHAR or READLIST/ READLISTCC. READCHAR will return only the first character that you type. READLIST and READLISTCC return a list, not a word. There are several ways to solve this problem. Perhaps the simplest is to write a couple of special procedures and save them as tools to be loaded whenever you need them.

Get a NEW PAGE from the Contents page and enter these procedures.

```
TO READWORD
OUTPUT FIRST READLIST
END

TO READWORDCC
OUTPUT FIRST READLISTCC
END
```

Save this page, perhaps as READWORD. Then, when you want to use these procedures on another page, you can type

```
LOAD "READWORD
```

LOAD adds procedures from the page READWORD to your current page. Don't worry about the "inner workings" of the READWORD and READWORDCC procedures for now. You will learn about OUTPUT in Chapter 31. FIRST will be discussed in Chapter 38 near the end of this book.

Using these procedures, you can now work with words and numbers. Here, for example, is a procedure for drawing designs interactively.

```
TO DRAW
(TYPE [How far forward?] CHAR 32)
FORWARD READWORDCC
(TYPE [How much right turn?] CHAR 32)
RIGHT READWORDCC
DRAW
END
```

If you type DRAW, you see

```
How far forward?
```

If you type

```
50
```

the turtle will move forward 50 steps. Then you see

```
How much right turn?
```

If you type

```
90
```

the turtle will turn right. Then the questions are repeated over and over until you press the Stop keys. This procedure makes a nice "drawing utility" for beginners.

Using READWORDCC you could create a very simple guessing game. For example

```
TO GUESS
PRINT []
INSERT [Guess a number:]
IF NOT (READWORDCC = 7) [GUESS]
END
```

When you type GUESS, the program asks you to guess a number. It continues to ask you to guess until you type 7. Notice the IF statement. The condition after IF is

```
NOT (READWORDCC = 7)
```

Recall that Logo checks to see if the number typed is equal to 7. If it is, then (READWORDCC = 7) is equal to "true." The NOT reverses that to "false" and Logo moves to the next line of the procedure, encounters END, and stops. If READWORDCC is not 7, then (READWORDCC = 7) is equal to "false" and the NOT reverses that to "true" and the procedure DRAW is called again.

TIPS AND TECHNIQUES

You already know how READCHAR works. Don't become overwhelmed by the new primitives in this chapter. There is very little difference between READCHAR and the new reporters in this chapter. They only extend the ideas you have already learned. READLIST/READLISTCC and READWORD/READWORDCC are simply "funnels" between the keyboard and Logo.

When you use READCHAR the cursor disappears and you can't see what is typed. When you use READLIST/ READLISTCC or READWORD/READWORDCC, the characters typed on the keyboard appear either on the page or in the Command Center.

Do not worry about understanding how READWORD or READWORDCC works at this point. They are reporters that really should be part of LogoWriter, but were omitted to save memory. Use them exactly as you would READCHAR or READLIST. Just remember to LOAD them onto your page when you want to use them.

When do you use READCHAR? When do you use READWORD or READWORDCC? When do you use READLIST or READLISTCC? The answers are really quite easy. Use READCHAR when you want to accept only one character from the keyboard. Use READWORD or READWORDCC when you want to accept a number (or perhaps a single word) from the keyboard. Use READLIST or READLISTCC in all other cases.

PROJECT SUGGESTION

Take some time to experiment with these interactive reporters. Write some simple interactive programs using READLIST, READLISTCC, READWORD, and READWORDCC. Be sure that you can use each one appropriately. Also, use the different forms of text output commands, PRINT, INSERT, SHOW, and TYPE. Practice making the text on the page or in the Command Center easy to read and understand.

24. Working With Pages

You have been working with pages since your first experiences with LogoWriter. You have created pages and saved them. You have gotten pages that you made earlier and worked with them. There are a number of LogoWriter commands for working with pages that you may not have encountered.

First, review the commands you already know.

- NAMEPAGE (NP) "pagename gives the current page a name.
- RESTORE replaces current page with the copy last saved using Esc.
- LOAD "pagename loads the procedures from "pagename" and adds them to the procedures on the current page.

In addition, remember that

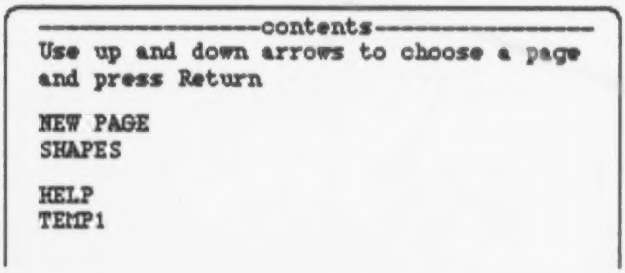
- Esc: saves the current page under the name at the top of the page

Take a few minutes to experiment with any of these that you have forgotten how to use.

Now for some new commands. Get a new page and name it TEMP1. Then, in the Command Center, type

```
CONTENTS
```

The current page (TEMP1) is saved and you see the Contents page.



```
----------------contents----------------
Use up and down arrows to choose a page
and press Return

NEW PAGE
SHAPES

HELP
TEMP1
```

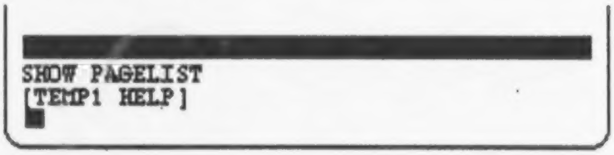
Select TEMP1. You see your newly created page. You can type CONTENTS rather than press Escape any time you want to go to the Contents page..

Sometimes you want to get another page, but you don't remember the names of the pages on your disk. You can type

```
PRINT PAGELIST
```

and the list of pages appears on the page, or

```
SHOW PAGELIST
```



```
SHOW PAGELIST
[TEMP1 HELP]
```

and a list of the pages appears in the Command Center.

PAGELIST is a reporter that returns the list of all the pages on the disk in the disk drive without leaving the current page and going to the Command Center.

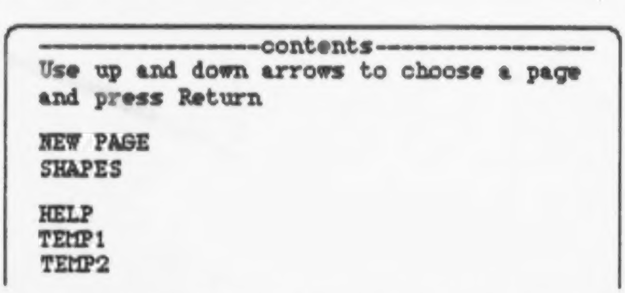
Sometimes you want to remove a page from the Contents page that you no longer want. There are two ways to do this. Go to the Contents page, move the cursor down to the page TEMP1, and press Erase to End of Line. The page is erased *permanently.*

Create the page TEMP1 again by getting a new page and naming it TEMP1. Press Escape to save it, get another new page and name it TEMP2. From the Command Center, type

```
ERPAGE "TEMP1
```

Go to the Contents page and you see the TEMP1 is gone.

Create page TEMP1 again. You should now have TEMP1 and TEMP2 on your Contents page.



```
----------------contents----------------
Use up and down arrows to choose a page
and press Return

NEW PAGE
SHAPES

HELP
TEMP1
TEMP2
```

Get page TEMP1 from the Contents page. Then, in the Command Center, type

```
GETPAGE "TEMP2
```

```
GP "TEMP2
```

Notice that TEMP1 is automatically saved, and TEMP2 is loaded in its place. TEMP2 is now the current page. You do not need to go to the Contents page to get another page.

When you are finished working with a page and want to get a new one, you do not have to return to the Contents page. You can use NEWPAGE. NEWPAGE saves the current page, and gets a new page. When you type

```
NEWPAGE
```

you see the ???'s at the top of the page. This is the same thing that happens when you get a new page from the Contents page.

You can also clear a page completely by using CP, Clear Page. This clears both sides of the page, thus removing all of the procedures from the back of the page. *Be very careful when using this command.* If you type

```
CP
```

and then press Esc, you save a completely blank page. This erases anything that was previously saved on the page, **including the procedures on the flip side of the page.** CP can be very dangerous!

There is even a command for flipping the page. So, instead of using the Flip keys, you can type

```
FLIP
```

in the Command Center. Notice, however, that when you type FLIP to get to the back of the page, the cursor is in the Command Center instead of on the page.

If you are using LogoWriter version 2.0 or above, then you can load shapes from any shapes page without leaving the current page. Simply type

```
GETSHAPES
```

in the Command Center, and the shape page from the disk in the disk drive is loaded into memory.

If you are using a more recent version of LogoWriter (2.0 or above) there are still more commands that help your work with pages. For example, the primitives LOADPIC and SAVEPIC allow you to load and save the front of a page. The primitives LOADTEXT and SAVETEXT allow you to load and save the text on a page. For more details on these and other useful primitives, see Appendix 6. These more recently added primitives also provide you with the capability of exchanging LogoWriter picture and text files with other software. For more details, see Appendix 7.

TIPS AND TECHNIQUES

Make an effort to learn to use the commands in this section. They can save you a lot of time. For example, getting a new page by typing NEWPAGE is much faster than pressing Esc to save the current page, returning to the Contents page, and then selecting NEWPAGE.

Any of the commands in this section can be used in procedures. Thus, you can be sure that the correct shapes page is in memory by including GETSHAPES in your SET.UP procedure. For example

```
TO SET.UP
GETSHAPES
CT
RG
END
```

PROJECT SUGGESTION

Take some time to experiment with these new commands and reporters. Will some of them make your work with LogoWriter simpler? Become comfortable with these new commands and reporters so you can easily use them in the future.

25. More on Making Decisions

You have used the IF command to write procedures that branch. You use IF when you want to make a decision within a procedure. Two things can happen when you use IF. Either the condition after the IF is true or it is false. If the condition is true, the list of commands in the IF statement are run. If the condition is false, Logo moves to the next instruction in the procedure.

```
                  [Do these commands]
IF   condition    [ if the condition ]
                  [     is true      ]

Otherwise, continue with the next command in the procedure
```

Have you ever wanted to write a procedure that causes Logo to take one action if the condition after the IF is true and another if the condition is false, and *then* go to the next statement in the procedure? The command that allows you to do that in LogoWriter is IFELSE.

IFELSE works like this:

```
                      [Do these commands] [Do these commands]
IFELSE condition      [if the condition   ] [if the condition   ]
                      [     is true       ] [     is false      ]
```

You have seen several examples using IF to ask the user whether or not to repeat some action. IFELSE makes it possible to make such a procedure more "user friendly."

```
TO SEE.DESIGN
DESIGN <—This is a procedure to draw some kind of graphics.
PRINT [Another design?]
IFELSE READCHAR = "Y
   [SEE.DESIGN] [PRINT [O.K. Goodbye, then!]]
END
```

If the user types Y, the design appears again. If any other key is typed, then the message

```
O.K. Goodbye, then!
```

appears.

Suppose you wanted to write a program to allow the user to choose from a menu. For example,

```
TO MENU
PRINT [1.See a picture]
PRINT [2.Answer a question]
IFELSE READWORD = 1 [PICTURE] [ASK.QUESTION]
END
```

Then, when you type

```
MENU
```

you can choose which action you want to take. Notice that if you pick 1, then procedure PICTURE is run. The only choice that makes the condition (READWORD = 1) true is the number 1. If any other key is pressed, (READWORD = 1) is false. Thus, pressing any key except the "1" key causes ASK.QUESTION to run. Even if the user types the wrong key, the question is asked anyway! You will learn how to avoid this problem in Chapter 28.

Did you note that READWORD is used in the procedure MENU? Recall that you have to write a procedure READWORD or READWORDCC in order to accept numbers from the keyboard.

```
TO READWORD
OUTPUT FIRST READLIST
END

TO READWORDCC
OUTPUT FIRST READLISTCC
END
```

You can use IFELSE to make various choices within a procedure. For example,

```
TO DESIGN
SETBG RANDOM 6
IFELSE (BG < 3) [DRAW.STAR] [DRAW.TRIANGLE]
SETSH 10 + RANDOM 21
IFELSE (SHAPE > 20) [MOVE.UP] [MOVE.RIGHT]
END
```

Here the design that appears on the screen is dependent on the numbers picked by RANDOM. First a random background color is picked. If the color number is 0, 1, or 2, then a star is drawn. If it is 3 or greater, then a triangle is drawn. Next a random shape number (between 10 and 30) is picked. If the number is bigger than 20 (SHAPE is a reporter that returns the number of the current shape), then the turtle is moved up; otherwise it is moved to the right.

Examine carefully what happens in a procedure with an IFELSE. When Logo encounters the IFELSE, it first tests the condition. If the condition is true, the first list after the condition is run. If the condition is false, the second list after the condition is run. Only after one of these lists is run does Logo continue with the next line of the procedure.

TIPS AND TECHNIQUES

In all of the examples in this chapter, there is one command in each of the lists after an IFELSE. There *can* be more than one command. For example

```
IFELSE (READWORD = 1) [FORWARD 50 RIGHT 90]
    [BACK 50 LEFT 90]
```

It is acceptable to put a few statements in each list after an IF or IFELSE. However, if those lists get very long, then the IF or IFELSE statement gets quite hard to read. It is better to write a separate procedure for each list. For example, the above statement might read

```
IFELSE (READWORD = 1) [GO.FD] [GO.BK]
```

where

```
TO GO.FD
FORWARD 50
RIGHT 90
END

TO GO.BK
BACK 50
LEFT 90
END
```

would be found on the flip side of the page.

Even a rather short IF or IFELSE statement can become hard to read if it wraps to the next line. It is better to use the wordprocessing capabilities of LogoWriter to format such statements. For example, the above IFELSE statement would be easier to read if you wrote

```
IFELSE (READWORD = 1)
    [FORWARD 50 RIGHT 90]
    [BACK 50 LEFT 90]
```

Keep in mind that Logo procedures are run one line at a time. That is, when Logo is running one line of a procedure, it does not "know" what the next line of a procedure will be. When you are debugging procedures, you might consider using an index card so that you can see only the line you are reading. Perhaps this technique will help you avoid the common error of assuming that the procedure does what you *want* it to do instead of what you *told* Logo to do.

When using an IF or an IFELSE to build a loop, be particularly careful that you don't accidentally introduce embedded recursion. Embedded recursion is a little harder to see when you are using an IFELSE. The recursive call must be the last statement to run. Consider this example.

```
TO EXAMPLE
PRINT [THIS IS AN DEMONSTRATION]
IFELSE (READCHAR = "Y)
    [EXAMPLE CT]
    [PRINT [BYE]]
END
```

This procedure has embedded recursion. Each time the user types "Y", EXAMPLE is run again, but the CT is not run. When the user types something other than "Y", the "PRINT [BYE]"is run. Although it would be very difficult to tell, the CT is then run as many times as the user typed "Y." (Try replacing the CT with a PRINT statement.) This problem can be corrected by changing the procedure as follows:

```
TO EXAMPLE
PRINT [THIS IS AN DEMONSTRATION]
IFELSE (READCHAR = "Y)
    [CT EXAMPLE]
    [PRINT [BYE]]
END
```

Now EXAMPLE is the last statement run if the user types "Y," and the procedure is now tail recursive.

PROJECT SUGGESTION

You have written programs using IF. Perhaps you wished for an IFELSE to add some action to your program. Go back to one of those programs and modify it using IFELSE to make it more powerful or interesting.

26. Procedure Inputs

Have you ever made a design out of similar polygons that differed only in size? For example, have you made a design made of squares or triangles of different size? Or perhaps, have you made a square house with square windows? If you needed squares of several sizes, then you had to write several square procedures that make the turtle go forward different amounts. Now you are going to learn how to use a single procedure to that can be used to draw a square of any size you wish.

To make polygons of different sizes, you use procedure inputs. Procedure inputs are names which stand for values that can be changed each time the procedure is used. For example, suppose you write

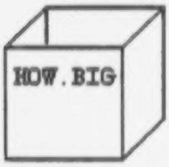
```
TO SQUARE :HOW.BIG
REPEAT 4 [FORWARD :HOW.BIG RIGHT 90]
END
```

If you type

```
SQUARE 50
```

then the name HOW.BIG has the value 50 associated with it. Then when FORWARD needs an input, it finds :HOW.BIG. The ":" (called "dots") in front of HOW.BIG mean to use the value associated with HOW.BIG.

Another way to think about this is to imagine a box that is created when SQUARE is defined. The box is empty and has the word HOW.BIG written on the outside.

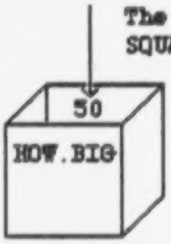


This "box" is created when SQUARE is defined.

Then, when you type

```
SQUARE 50
```

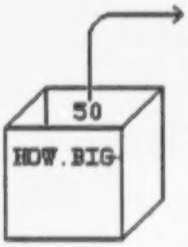
the number 50 is put in the box.



The number 50, from SQUARE 50, is put into the box.

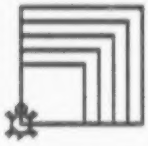
Later, when FORWARD finds :HOW.BIG for its input, Logo looks in the HOW.BIG box to get the value to give FORWARD as input.

You can now use the SQUARE procedure listed earlier to make designs. For example,



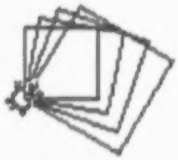
A copy of the 50 is then used by FORWARD to tell Logo how far to move the turtle.

```
TO DESIGN1
SQUARE 20
SQUARE 25
SQUARE 30
SQUARE 35
SQUARE 40
END
```



or

```
TO DESIGN2
SQUARE 30
RIGHT 10
SQUARE 35
RIGHT 10
SQUARE 40
RIGHT 10
SQUARE 45
END
```



Whenever a procedure is written using a procedure input, you must always provide the value for the input name when using that procedure. For example, if you type

```
SQUARE
```

and press Return/Enter, Logo will say

```
SQUARE needs more inputs
```

just as it does when you type FORWARD without an input. Procedures that you write that use inputs behave exactly like Logo primitives that use inputs.

LOOPING USING INPUTS

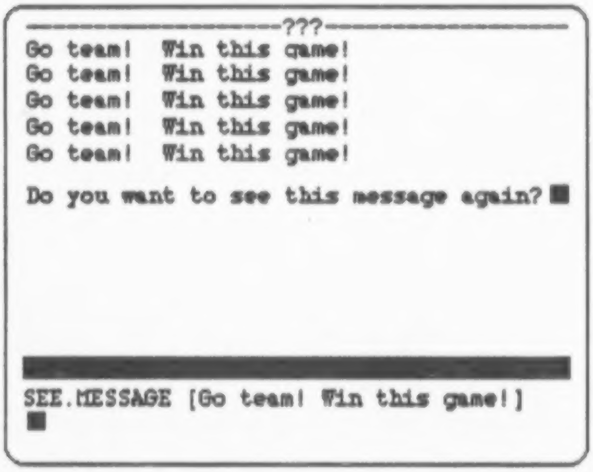
You have written procedures that ask the user if s/he wants to see a message again. Using procedure inputs, you can put any message you want in such a procedure. For example,

```
TO SEE.MESSAGE :MESSAGE
CT
REPEAT 5 [PRINT :MESSAGE]
PRINT []
INSERT [Do you want to see the message
    again?]
IF READCHAR = "Y [SEE.MESSAGE :MESSAGE]
END
```

To run this procedure, type

```
SEE.MESSAGE [Go team! Win this game!]
```

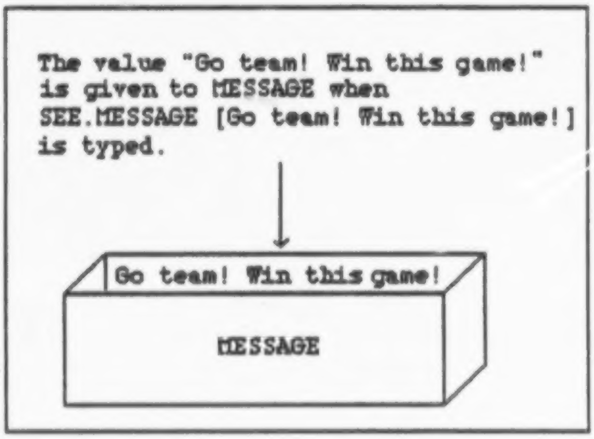
You see



```
-------------------???-------------------
Go team!  Win this game!
Go team!  Win this game!
Go team!  Win this game!
Go team!  Win this game!
Go team!  Win this game!
Do you want to see this message again? ■



SEE.MESSAGE [Go team! Win this game!]
■
```

If you type Y, then the lines of the procedure are repeated. Do you see what is happening?

- The words "Go team! Win this game!" are put in the "box" labeled "MESSAGE".



The value "Go team! Win this game!" is given to MESSAGE when SEE.MESSAGE [Go team! Win this game!] is typed.

- Next PRINT has as input :MESSAGE, read "dots MESSAGE." (The ":" symbol can be thought of as "look in the box.")
- PRINT finds "Go team! Win this game!" in the "MESSAGE box" and prints that value 5 times on the page.
- Next, the READCHAR causes the procedure to wait for input. If Y is typed, then the "value in the MESSAGE box" is sent back to procedure SEE.MESSAGE, using the name MESSAGE.

You can make this procedure more user friendly by using IFELSE:

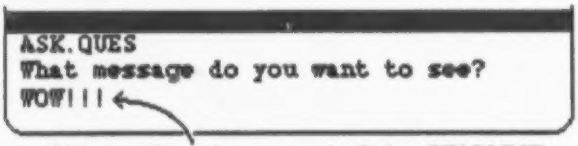
```
TO SEE.MESSAGE :MESSAGE
CT
REPEAT 15 [PRINT :MESSAGE]
PRINT []
INSERT [Do you want to see the message
    again?]
IFELSE READCHAR = "Y
      [SEE.MESSAGE :MESSAGE]
      [PRINT [O.K., goodbye!]]
END
```

PASSING VALUES FROM THE KEYBOARD TO A PROCEDURE INPUT

Adding one more procedure makes the program even more user friendly.

```
TO ASK.QUES
PRINT [What message do you want to see?]
SEE.MESSAGE READLIST
END
```

Now, whatever is reported by READLIST is put in the MESSAGE box when SEE.MESSAGE is called

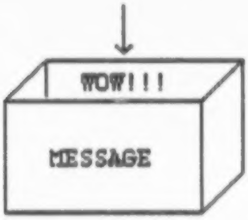


```
ASK.QUES
What message do you want to see?
WOW!!!
```

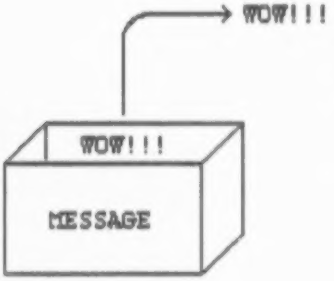
This value is accepted by READLIST.

"WOW!!!" is reported to SEE.MESSAGE.

"WOW!!!" becomes the value of MESSAGE.



and is then used by the procedure SEE.MESSAGE.



WOW!!! becomes the input to PRINT and is displayed 5 times on the page.

This last example contains a lot of new ideas.

- a non-numeric value is associated with an input
- the value of the input is "passed back" to the procedure using the input name
- a value is "passed to" the input name using READLIST.

You will learn more about these ideas in coming chapters. For now, take a few minutes to study the example carefully. You will likely need to read this example several times before you begin to understand it.

CREATING A CASE STATEMENT

Earlier you encountered the problem of creating a menu with two items using IFELSE. If you run the procedure MENU below, all is well if you select either "1" or "2", but if you type any other number, the ASK.QUESTION procedure runs, even though that is probably not what you want to happen.

```
TO MENU
PRINT [1.See a picture]
PRINT [2.Answer a question]
IFELSE READWORD = 1 [PICTURE] [ASK.QUESTION]
END
```

Sometimes you want a procedure to allow multiple choices and, in addition, to do nothing at all if an incorrect choice is made. To accomplish this task, you must write a more elaborate program. Suppose you have the following menu procedure.

```
TO SHOW.CHOICES
PRINT [1. See a square]
PRINT [2. See a star]
PRINT [3. See a triangle]
END
```

You can then create a procedure to handle all of the choices. This type of structure is called a CASE statement.

```
TO WHICH.ONE :CHOICE
IF :CHOICE = 1 [SQUARE STOP]
IF :CHOICE = 2 [STAR STOP]
IF :CHOICE = 3 [TRIANGLE STOP]
END
```

Now you can write

```
TO MENU
SHOW.CHOICES
WHICH.ONE READCHAR
END
```

Think about how this procedure works.

- First the menu is displayed on the page.
- Then the procedure WHICH.ONE is called.
- READCHAR waits for input from the keyboard and passes the valued typed into the name CHOICE in procedure WHICH.ONE.
- CHOICE is then tested by each IF statement. If there is a match, then the appropriate list of commands is run.

The STOP statement is not usually necessary in this case structure. However, if you have a lot of choices, it prevents the procedure from running through all possible matches listed in the procedure. STOP causes only the procedure is which it is located to stop running.

TIPS AND TECHNIQUES

There are a lot of new ideas in this chapter. If the idea of procedure inputs is new to you, then start with simple procedures like the ones for drawing polygons. Only when you understand these simple procedures should you try to write procedures with inputs that use interactive reporters and/or conditionals (IF and IFELSE).

The model used for procedure inputs in this chapter is boxes containing values. The value of this model is that it is easy to draw and explain. However, it has a serious flaw. A Logo name can be associated with only one value. That is, the "box" can contain only one value. However, boxes in the real world can contain more than one item. Further, when you pass a value to another name, it is still associated with the original name. A *copy* is passed. But when you take something out of a real world box, it is gone. Be careful that you don't take the model of values put into boxes to literally. Keep in mind that a Logo name can hold only one value and that it passes a copy of that value to another name.

Using PRINT statements in procedures which use inputs can be very helpful in debugging. For example, you can use a PRINT statement to verify the value of an input.

```
TO SQUARE :HOW.BIG
(SHOW [THE VALUE OF "HOW BIG IS] :HOW.BIG)
REPEAT 4 [FORWARD :HOW.BIG RIGHT 90]
END
```

You can keep track of how often a procedure is run with a given input

```
TO SEE.MESSAGE :MESSAGE
(SHOW [BEGINNING OF SEE.MESSAGE WITH
   VALUE] :MESSAGE)
CT
REPEAT 5 [PRINT :MESSAGE]
PRINT []
INSERT [Do you want to see the message
   again?]
IF READCHAR = "Y [SEE.MESSAGE :MESSAGE]
END
```

or verify what value was passed to a procedure.

```
TO WHICH.ONE :CHOICE
(PRINT [THE VALUE PASSED TO CHOICE IS ]
   :CHOICE)
IF :CHOICE = 1 [SQUARE STOP]
IF :CHOICE = 2 [STAR STOP]
IF :CHOICE = 3 [TRIANGLE STOP]
END
```

Using PRINT or SHOW to trace the action of your procedures can save you a lot of debugging time. This is a case where you should let Logo do some of the debugging work for you.

PROJECT SUGGESTIONS

Write procedures to draw a design made up of squares and triangles of all sizes. Be sure that you use an input and write only one SQUARE and one TRIANGLE procedure.

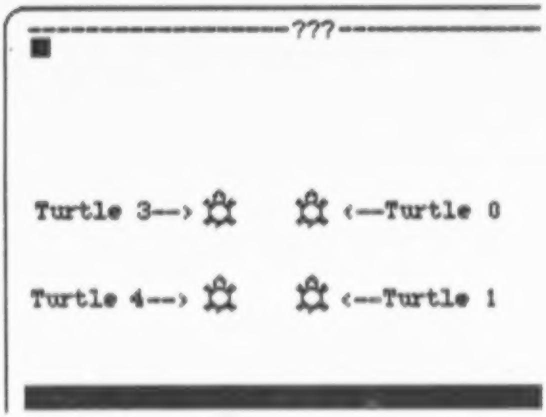
Write a program that asks the user what size design s/he wants to see. Then draw the design using what the user types as input.

Write a procedure with more than two menu choices. Are you sure all of the possibilities work as you intend? Test your program carefully.

Think about a procedure to draw a triangle. You use REPEAT 3 followed by some commands. To draw a square, you use REPEAT 4. What would you use to draw a 5 sided figure? Can you write a procedure to draw a figure with any number of sides using an input? What angle will you turn? (Hint: The turtle must turn 360 degrees to get back to where it started.)

27. More on Multiple Turtles

You have used multiple turtles. Recall that you can have several turtles active at once, all doing the same thing. The turtles are numbered 0, 1, 2, and 3.



You saw that

```
TELL number or list of numbers
```

tells which turtle or turtles should become active.

There are several other LogoWriter primitives that are used with multiple turtles. Sometimes when you are working with more than one turtle, you forget which turtles are active. That problem is easily solved with WHO. Recall that WHO is a reporter that returns the numbers of the active turtles. Thus, if you type

```
TELL [1 3]
```

and then later in your work you forget which turtles are active, you can type

```
SHOW WHO
```

and LogoWriter will display

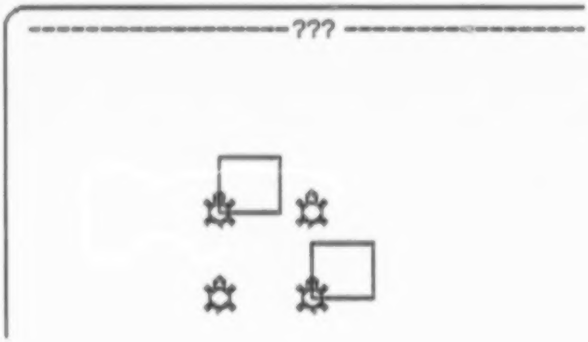
```
[1 3]
```

in the Command Center.

Another useful command is EACH. EACH is followed by a list of commands for the active turtles to run. If you type

```
TELL [3 1]
SQUARE 20
```

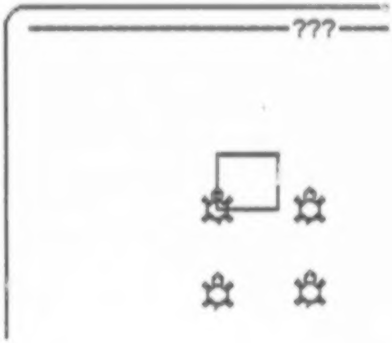
then turtles 3 and 1 draw a square *at the same time* (assuming you have defined a procedure SQUARE.) You see



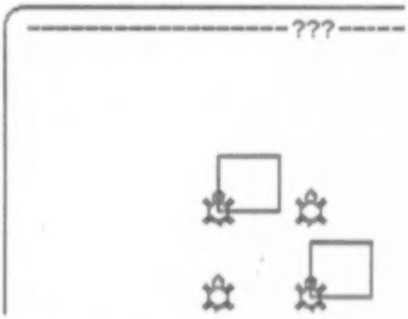
However, if you then type

```
EACH [SQUARE 20]
```

you see



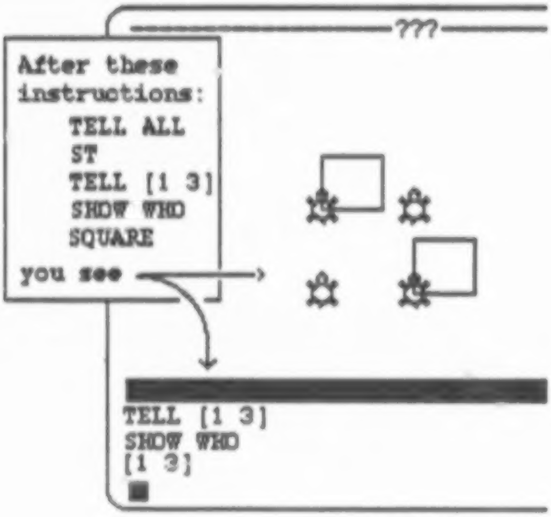
followed by



Turtle 3 draws a square, and then turtle 1 draws a square. Notice the difference. If you issue commands after a TELL, all of the active turtles move at once. If you use EACH followed by a list of commands, the turtles move one at a time.

A third useful command is ASK. ASK needs two inputs: a turtle number or list of turtle numbers, and a list of commands. It allows you to address either an active or an inactive turtle. Here is a list of commands to illustrate these ideas.

```
TELL ALL
ST
TELL [1 3]
SHOW WHO
SQUARE 20
ASK 2 [BACK 20]
SHOW WHO
ASK 1 [FORWARD 20]
SHOW WHO
```



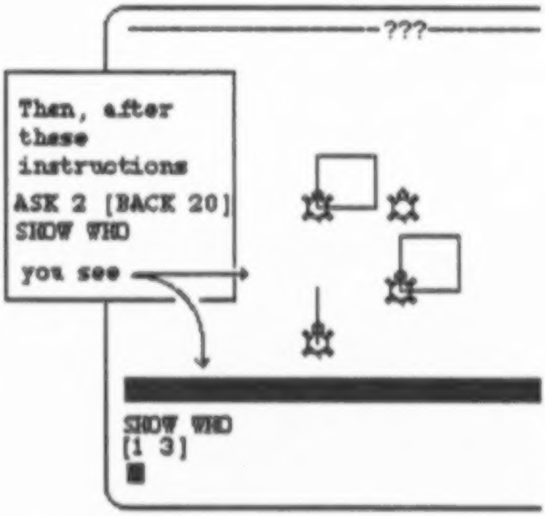
After these instructions:

```
    TELL ALL
    ST
    TELL [1 3]
    SHOW WHO
    SQUARE
```

you see

```
TELL [1 3]
SHOW WHO
[1 3]
```

Turtles 1 and 3 draw a square because they are made active by TELL.



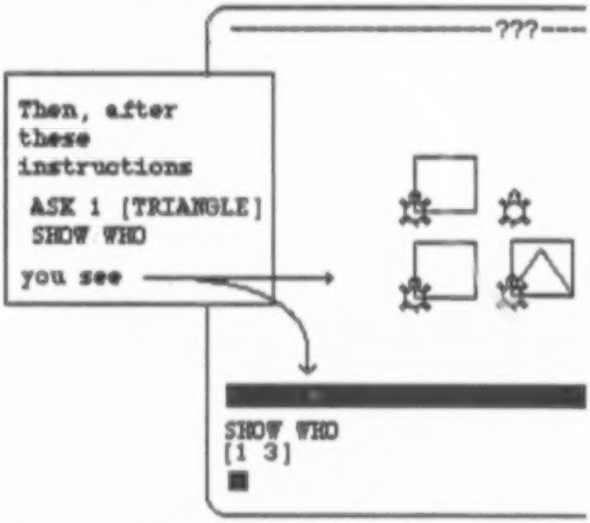
Then, after these instructions

```
ASK 2 [BACK 20]
SHOW WHO
```

you see

```
SHOW WHO
[1 3]
```

ASK communicates with turtle 2, and instructs it to move back 20 turtle steps. However, the list of active turtles is unchanged. Only TELL changes the list of active turtles that is reported by WHO .



Then, after these instructions

```
ASK 1 [TRIANGLE]
SHOW WHO
```

you see

```
SHOW WHO
[1 3]
```

This time ASK causes the second turtle to draw a triangle, but the active list is not changed. Take a few moments to study this example to be sure you understand what is happening.

Using these new commands and interactive commands, you can write an interactive program to make the turtles dance. First, you need a dance procedure.

```
TO DANCE
REPEAT 20 [FORWARD 1 LEFT 18 WAIT 1]
REPEAT 20 [BACK 1 RIGHT 18 WAIT 1]
REPEAT 10 [FORWARD 1 WAIT 2]
REPEAT 10 [BACK 1 WAIT 2]
REPEAT 20 [FORWARD 1 RIGHT 18 WAIT 1]
REPEAT 20 [BACK 1 LEFT 18 WAIT 1]
END
```

Next, you need to place the turtles in a row on the screen.

```
TO SET.UP
RG
CT
TELL ALL
PU
ST
TELL 1
FORWARD 40
RIGHT 90
FORWARD 40
LEFT 90
TELL 2
FORWARD 40
LEFT 90
FORWARD 40
RIGHT 90
END
```

Now for the main procedure

```
TO ASK.QUES
SET.UP
ASK.USER
END
```

and the ASK.USER procedure.

```
TO ASK.USER
PRINT []
INSERT [Which turtles do you want to do a
   dance?]
TELL READLIST
PRINT []
INSERT [All at once or separately?  (A=All
   at once  S=Separately)]
IFELSE READCHAR = "A [DANCE] [EACH [DANCE]]
PRINT []
INSERT [Which turtle would you like to see
   dance a solo?]
ASK READCHAR [DANCE]
PRINT []
INSERT [Would you like to see some more
   dancing?]
IF READCHAR = "Y [ASK.USER]
END
```

Using a procedure input, you could even change the speed of the dance.

```
TO DANCE :SPEED
REPEAT 20 [FORWARD 1 LEFT 18 WAIT :SPEED]
REPEAT 20 [BACK 1 RIGHT 18 WAIT :SPEED]
REPEAT 10 [FORWARD 1 WAIT 2 + :SPEED]
REPEAT 10 [BK 1 WAIT 2 + :SPEED]
REPEAT 20 [FORWARD 1 RIGHT 18 WAIT :SPEED]
REPEAT 20 [BACK 1 LEFT 18 WAIT :SPEED]
END
```

Then, you could replace each DANCE in ASK.USER with ASK.DANCE

```
TO ASK.DANCE
INSERT [How fast should the dance be?]
DANCE READCHAR
PRINT []
END
```

What other modifications might you make to this program? How about asking the user what color each turtle should be? How about asking about the background color?

TIPS AND TECHNIQUES

The differences among TELL, EACH, and ASK can be confusing at first. Here is a brief summary to help you understand each command.

- TELL sets up the list of actives turtles.
- EACH causes the list of active turtles to perform some action
- ASK ignores the list of active turtles and directs any turtle or turtles to perform some action.

Why do you need each of these statements?

- TELL is used to change the list of active turtles. It has no other function. Turtle commands given *after* a TELL command cause all of the turtles to do those commands *at the same time*.
- EACH is used if you want the turtles to perform some action *one at a time*.
- ASK is used to give direction to turtles *not in the active list*. For example, you might "ASK" a non-active turtle to return its color, shape, or heading.

The ASK.USER procedure in this chapter combines a lot of ideas that you have learned in previous chapters. It is also a good example of the kind of procedure where inserting PRINT or SHOW statements can help in debugging. Here is the ASK.USER procedure with a number of possible diagnostic statements inserted that might help in the debugging.

```
TO ASK.USER
SHOW [AT THE BEGINNING OF ASK.USER]
PRINT []
INSERT [Which turtles do you want to do a
   dance?]
TELL READLIST
SHOW WHO   <-This shows which turtles are now active
PRINT []
INSERT [All at once or separately?  (A=All
   at once  S=Separately)]
IFELSE READCHAR = "A
   [SHOW [A SELECTED] DANCE]
   [SHOW [A NOT SELECTED] EACH [DANCE]]
PRINT []
INSERT [Which turtle would you like to see
   dance a solo?]
ASK READCHAR [(SHOW [TURTLE PICKED IS] WHO)
   DANCE]  <-WHO used inside the ASK statement
PRINT []       returns the turtle active at that moment.
INSERT [Would you like to see some more
   dancing?]
IF READCHAR = "Y [ASK.USER]
END
```

Take a few moments to examine where the diagnostic statements have been added. Learning to put such statements into useful places takes practice and is part of the art of debugging.

PROJECT SUGGESTION

Write an interactive program using multiple turtles and inputs. Perhaps you could have the turtle draw geometric shapes or draw different colored designs. Start with a simple idea and expand it as you get more ideas.

28. Using Inputs to Pass Values Among Procedures

You now know how to write procedures with inputs. You can write a single procedure to draw squares of any size or print a variety of messages. You also know how to send a value typed by a user to a procedure with an input. However, what if you want to send a value typed by the user to several procedures? For example, suppose you have a procedure that asks the user to tell how big s/he wants some polygons or other pictures. You can't write

```
TO ASK.QUES
PRINT [What sized polygons?]
SQUARE READWORD      <--Remember, you have to define
TRIANGLE READWORD             READWORD.
END
```

Do you see why? If you type

```
ASK.QUES
```

Logo puts

```
What sized polygons?
```

on the page. If the user then types

```
50
```

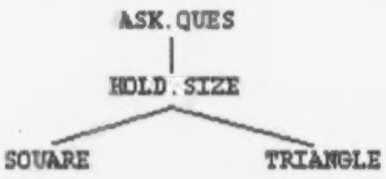
then a square of size 50 is drawn. Everything seems fine so far.

Next, the procedure stops and waits for a second input because it encounters another READWORD. But you wanted a square and triangle of the same size. The user should not have to type 50 again.

Here is one way to solve this problem.

```
TO ASK.QUES
PRINT [What sized polygons?]
HOLD.SIZE READWORD
END

TO HOLD.SIZE :SIZE
SQUARE :SIZE
TRIANGLE :SIZE
END

TO SQUARE :HOW.BIG
REPEAT 4 [FORWARD :HOW.BIG RIGHT 90]
END

TO TRIANGLE :LENGTH
REPEAT 3 [FORWARD :LENGTH RIGHT 120]
END
```

Do you see how HOLD.SIZE works? Examine the procedure tree to help you see the structure of the program. Notice that procedure inputs do not appear on the procedure tree.



Carefully study the following explanation of how ASK.QUES works. Take special note that the same input name is not used in any two procedures. While it is legal to use the same name, keeping the names different helps you to think about what is happening to the values of the names as the program runs. It also helps you avoid future bugs in more complex programs.

If you type

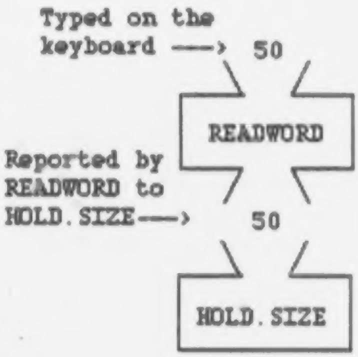
```
ASK.QUES
```

Logo responds with

```
What sized polygons?
```

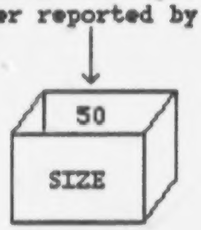
Suppose you type

```
50
```


Typed on the keyboard ---> 50
READWORD
Reported by READWORD to HOLD.SIZE---> 50
HOLD.SIZE

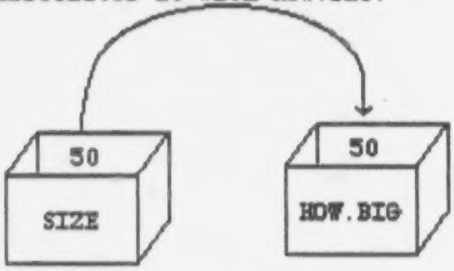
Then READWORD reports 50 to HOLD.SIZE. HOLD.SIZE needs an input. 50 becomes the input to the HOLD.SIZE procedure. Thus, SIZE has the value of 50 in HOLD.SIZE.

The value of the input to HOLD.SIZE gets the number reported by READWORD.



Next, SQUARE is called in the procedure HOLD.VALUE. SQUARE needs an input. The ":SIZE" is read "the value in the box SIZE." Logo then "passes" the value associated with SIZE into the HOW.BIG box.

Logo takes the value of SIZE and associates it with HOW.BIG.

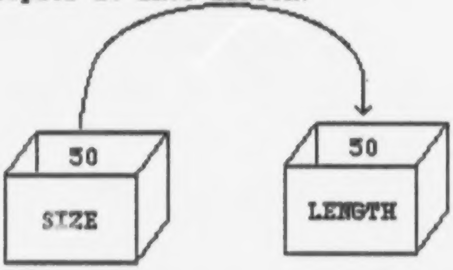


In the procedure SQUARE, the instruction

```
FORWARD :HOW.BIG
```

instructs Logo to look for the "value in the box HOW.BIG" to see how far forward to move the turtle.

Logo takes the value of SIZE and copies it into LENGTH.



Finally, the procedure TRIANGLE is called. In a similar manner, Logo copies the value of HOW.BIG into the LENGTH box. Now the value of LENGTH is used to move the turtle forward to draw a triangle.

The above explanation details a rather complex process called parameter passing by computer scientists. If you didn't understand it completely, don't be concerned. As you work with this idea, it will be become clearer simply by modeling it and through practice. You should read the explanation several times, and return to it in the future when you encounter similar examples. When you can draw the diagrams yourself and explain what is happening, you will have made this idea your own. You will then be able to use it in a variety of situations.

Here is another example. Can you explain what is happening at each step of the process?

```
TO GREET
PRINT [What is your name?]
HOLD.NAME READLIST
END

TO HOLD.NAME :NAME
PRINT.GREETING :NAME
SHOW.DESIGN
SAY.FAREWELL :NAME
END

TO PRINT.GREETING :WHO
(PRINT [Hi,] :WHO)
PRINT [Here is a neat design.]
END

TO SHOW.DESIGN
 (some graphics here)
END

TO SAY.FAREWELL :TO.WHOM
(PRINT [So long, ] :TO.WHOM)
PRINT [Hope you enjoyed the graphics.]
END
```

Did you conclude that READWORD passes a value to NAME in HOLD.NAME? Then the value of NAME is copied into WHO in PRINT.GREETING. Finally, the value of NAME is copied into TO.WHOM to be used in the procedure SAY.FAREWELL.

Notice that the name of the user is used in the first and last procedure in HOLD.NAME, but not in the design procedure. This technique allows you to use a piece of information anywhere you want in a program.

TIPS AND TECHNIQUES

The caution about the "box" model in Chapter 26 applies here as well. A Logo name can hold only one value at a time. When a value is passed to another name, a copy is passed; the original name keeps the value as well.

When you begin to use procedure inputs in a number of procedures in your program, it is important that you check each procedure carefully to be sure that it runs by itself. For example, you can check the HOLD.NAME procedure

```
TO HOLD.NAME :NAME
PRINT.GREETING :NAME
SHOW.DESIGN
SAY.FAREWELL :NAME
END
```

by typing something like

```
HOLD.NAME "MARY
```

and you can check the PRINT.GREETING procedure

```
TO PRINT.GREETING :WHO
(PRINT [Hi,] :WHO)
PRINT [Here is a neat design.]
END
```

by typing

```
PRINT.GREETING "MARY
```

If each procedure doesn't run by itself, then you are probably using procedure inputs wrong.

You should have noticed that a procedure input name is never used in more than one procedure. This techniques helps you avoid bugs as you learn to use procedure inputs. It forces you to be sure that each procedure works by itself without being dependent on any other procedure in the program to run correctly.

PROJECT SUGGESTIONS

Draw the diagrams and "boxes" to explain the program GREET.

Practice writing programs that use a value typed at the keyboard. At first, use the models above, but then see if you can develop a program that goes beyond the models given in this chapter.

29. Incrementing

Have you ever wanted to write a program that counted something? Perhaps you wanted the user to say how many times s/he wanted a message to appear. Or perhaps, you wanted to write a program to see all of the shapes on the Shapes page? You can write programs that count by building loops using tail recursion. Begin with this procedure

```
TO COUNT.UP :INPUT.VALUE
PRINT :INPUT.VALUE
WAIT 5
COUNT.UP :INPUT.VALUE + 1
END
```

If you type

```
COUNT.UP 1
```

then Logo responds with

```
1
2
3
4
5
.
.
.
```

forever! There is no stop rule. Before thinking about how to stop this procedure, examine carefully what is happening.

```
Begin with a value of 1 for INPUT, and substitute values:

TO COUNT.UP 1
PRINT 1
WAIT 5
COUNT.UP 1 + 1 —>TO COUNT.UP 2
END               PRINT 2
                  WAIT 5
                  COUNT.UP 2 + 1 —> TO COUNT.UP 3
                  END               PRINT :INPUT
                                    WAIT 5
                                    COUNT.UP 3 + 1 --> and so forth
                                    END
```

To stop the procedure at 10, you could add an IF instruction.

```
TO COUNT.UP :INPUT.VALUE
PRINT :INPUT.VALUE
WAIT 5
IF :INPUT.VALUE < 10 [COUNT.UP :INPUT.VALUE + 1]
END
```

Now the procedure will add one to the input value until it reaches 10. Examine the last several procedure calls.

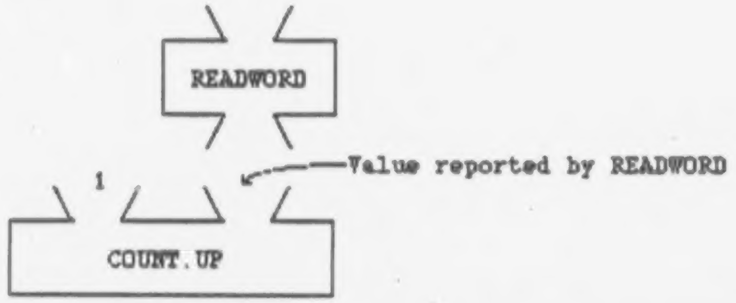
```
                Begin with INPUT.VALUE having the value of 8.
TO COUNT.UP 8
PRINT 8
WAIT 5
IF 8 < 10 [COUNT.UP 8 + 1]  (8<10 is true, so COUNT.UP
END                           is called with 9)
        TO COUNT.UP 9
        PRINT 9
        WAIT 5
        IF 9 < 10 [COUNT.UP 9 + 1]  (9<10 is true, so COUNT.UP
        END                           is called with 10)
                TO COUNT.UP 10
                PRINT 10
                WAIT 5
                IF 10 < 10 [COUNT.UP 10 + 1] (10<10 is false, so the
                END                           list is ignored, and
                                              Logo sees the END and
                                              stops the procedure.
```

This procedure can now be used in a program that asks the user how high s/he wants to count.

```
TO ASK.COUNT
PRINT [How high do you want to count?]
COUNT.UP 1 READWORD
END

TO COUNT.UP :INPUT.VALUE :MAX
PRINT :INPUT.VALUE
WAIT 5
IF :INPUT.VALUE < :MAX [COUNT.UP (:INPUT.VALUE + 1) :MAX]
END
```

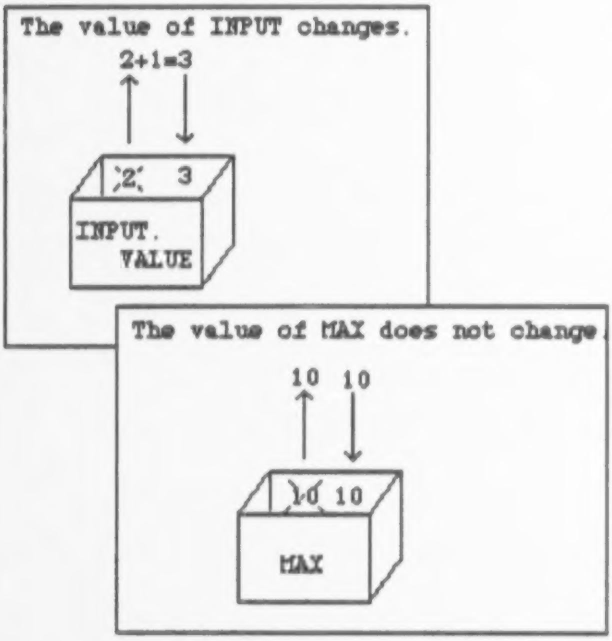
Notice that the procedure now has two inputs: the number that is printed in the procedure and the maximum.

COUNT.UP needs two inputs:



When COUNT.UP is called in ASK.COUNT, the number 1 is put into the "INPUT.VALUE box." The number typed by the user (and reported by READWORD) is put into the "MAX box." At the end of the procedure, the IF statement checks to see if the number input is less than the maximum. If so, then the procedure is called recursively.

Look carefully at this recursive call. The first value sent back to COUNT.UP is :INPUT.VALUE + 1. That is, one is added to the initial input so that when the procedure runs again, it runs with a number one larger than the original input. The value of MAX is passed unchanged.


The value of INPUT changes.
2+1=3
2 3
INPUT.
VALUE
The value of MAX does not change.
10 10
10 10
MAX

Note that in the stop rule, (:INPUT.VALUE + 1) is in parentheses. The parentheses are not necessary in this case. They are only there to make the procedure more readable.

The programs shown so far count by 1's — that is, 1 is added each time the procedure is called. What if you want to count by 5's or 10's? Instead of adding 1 each time, you can add 5 or 10. COUNT.UP can be modified as follows to count by 5's:

```
TO COUNT.UP :INPUT.VALUE :MAX
PRINT :INPUT.VALUE
WAIT 5
IF :INPUT.VALUE < :MAX
   [COUNT.UP (:INPUT.VALUE + 5) :MAX]
END
```

Now, instead of adding 1 each time, 5 is added. Can you draw diagrams like those given above to show what is happening to the values of each procedure input?

It is possible to generalize COUNT.UP even further by including yet another procedure input for the amount to add. This procedure becomes even more difficult to follow, but it provides a good exercise for you to see if you understand procedures with multiple inputs and passing values. Here the input TO.ADD holds the value to be added to INPUT.VALUE each time.

```
TO COUNT.UP :INPUT.VALUE :TO.ADD :MAX
PRINT :INPUT.VALUE
WAIT 5
IF :INPUT.VALUE < :MAX
   [COUNT.UP (:INPUT.VALUE + :TO.ADD)
   :TO.ADD :MAX]
END
```

You can take this one step further by allowing the user to input the amount to add.

```
TO ASK.COUNT
PRINT [Type the amount you want to]
PRINT [add followed by Return.]
PRINT [Then type the highest number you]
PRINT [want the program to print, and]
PRINT [press Return.]
COUNT.UP 7 READWORD READWORD
END
```

Think carefully about what is happening. Now there are three inputs about which you need to be concerned. If you type

```
ASK.COUNT
```

Logo responds with

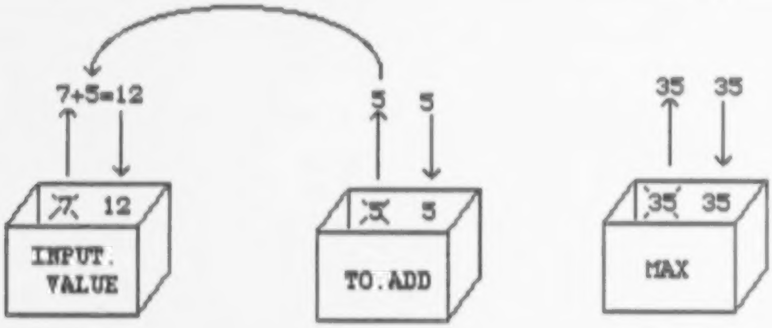
```
Type the amount you want to
add followed by Return.
Then type the highest number you
want the program to print, and
press Return.
```

Notice that you must type Return/Enter for each READWORD. READWORD accepts the first word in a list that is entered and ended with a Return/Enter.

If you then type

```
5
35
```

COUNT.UP is called with a value of 7 for INPUT.VALUE, 5 for TO.ADD, and 35 for MAX. See the diagram at the top of the next page.



The values of TO.ADD and MAX do not change.

The value of INPUT.VALUE is increased by the value of TO.ADD each time the procedure is called.

Can you predict what the output from this program will look like? There is a bug in this program. It does not perform exactly as described. Spend some time thinking through each recursive call to the procedure. Can you find the bug?

One further modification of the procedure allows the user to input the number to start with, namely the value of INPUT.VALUE.

```
TO ASK.COUNT
PRINT [Type the number you want to]
PRINT [start with followed by Return.]
PRINT [Then type the amount you want to]
PRINT [add followed by Return.]
PRINT [Then type the highest number you]
PRINT [want the program to print, and]
PRINT [press Return.]
COUNT.UP READWORD READWORD READWORD
END

TO COUNT.UP :INPUT.VALUE :TO.ADD :MAX
PRINT :INPUT.VALUE
WAIT 5
IF :INPUT.VALUE < :MAX
    [COUNT.UP (:INPUT.VALUE + :TO.ADD)
     :TO.ADD :MAX]
END
```

Can you draw the diagrams to explain this procedure?

Notice that the value reported by the first READWORD is put in the "INPUT.VALUE box;" the value reported by the second READWORD is put in the "TO.ADD box"; and the value reported by the third READWORD is put in the "MAX box." Both the MAX and the TO.ADD boxes remain unchanged as the procedure runs. As before, only the INPUT.VALUE box changes as the amount to add each time is added.

Now the program is even more flexible. Try running it. This version has the same bug. It doesn't count as high as it says it does. Can you fix this bug?

USING INCREMENTING

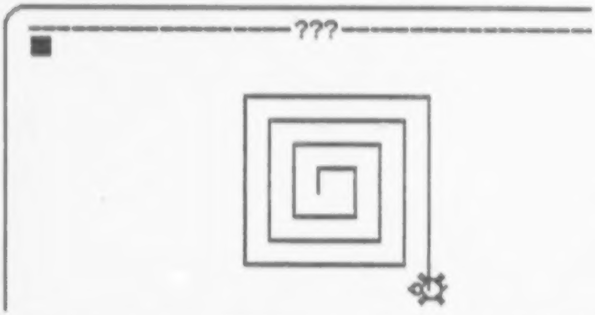
Incrementing can be used in a variety of settings. Suppose you want to make a "square spiral," that is, a square that gets longer as each side is drawn. You need to add a fixed amount each time you draw a side. This is similar to the COUNT.UP program above. You could write

```
TO GROW.SQUARE :SIDE
FORWARD :SIDE
RIGHT 90
IF :SIDE < 80 [GROW.SQUARE :SIDE + 5]
END
```

If you type

```
GROW.SQUARE 10
```

the turtle draws a side of length 10, then a side of length 15, then a side of length 20, and so forth until the side reaches 80.



How about writing a program to display all of the shapes? Recall that

```
SETSH 10
```

sets the turtle shape to shape number 10. Sometimes it would be helpful to see your shapes without leaving your current page. You can write a procedure that steps through the shapes one at a time. You can start with shape number 1 and add 1 until you reach your last shape number, say 30. This is similar to the COUNT.UP procedure.

```
TO SEE.SHAPES :WHICH.ONE
SETSH :WHICH.ONE
WAIT 10
IF :WHICH.ONE < 30 [SEE.SHAPES :WHICH.ONE
   + 1]
END
```

When you type

```
SEE.SHAPES 1
```

you see each shape on the screen briefly.

You can also use incrementing to count how often something happens. For example,

```
TO DRAW.STARS :COUNTER :HOW.MANY
PLACE.STAR   <— a procedure to pick a random location
STAR         <— a procedure to draw a star
IFELSE :COUNTER < :HOW.MANY
     [DRAW.STARS (:COUNTER + 1)  :HOW.MANY]
     [(PRINT [You see] :HOW.MANY [stars!])]
END
```

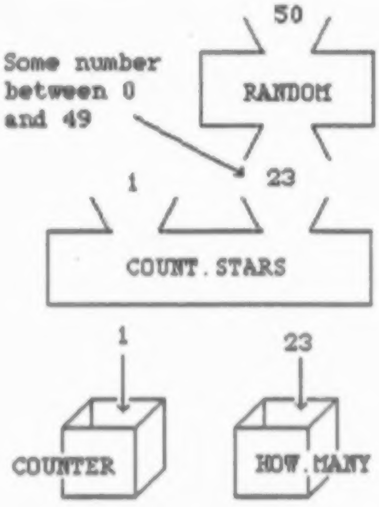
If you type

```
DRAW.STARS 1 20
```

This procedure draws 20 stars and then prints on the page that there are, indeed, 20 stars. This seems a bit strange. The program simply tells you that you have the number of stars that you asked for. You could use this procedure in a different way. By giving the input HOW.MANY a random number as input, each time the procedure is run a different number of stars are drawn. Then, printing the number drawn doesn't seem so strange.

```
DRAW.STARS 1 (RANDOM 50)
```

In this procedure, RANDOM 50 provides the input for HOW.MANY and COUNTER keeps track of the number of stars drawn.



Incrementing is a powerful tool. You can find many uses for it in your programs. Whenever you want to keep track of the number of times something occurs or you want to repeatedly add the same number then you need to use the techniques explained in this chapter.

EMBEDDED RECURSION

When you first learned to construct loops in Logo using recursive calls, you were cautioned to be sure that the recursive call was the last statement in your program or procedure. You saw an example in Chapter 19 in which the last lines of a procedure were never run because the procedure simply called itself again and again. In this context, embedded recursion looks like it has no particular value at all. It is, however, a powerful tool that is used extensively in more advanced Logo programming.

"Counting" procedures provide a good opportunity to further examine embedded recursion. Consider a modification of the simplest COUNT.UP procedure from the beginning of this chapter, renamed now as DO.COUNT.

```
TO DO.COUNT :INPUT.VALUE
PRINT :INPUT.VALUE
DO.COUNT :INPUT.VALUE + 1
END
```

Next, modify this procedure so that it includes a stop rule.

```
TO DO.COUNT :INPUT.VALUE
PRINT :INPUT.VALUE
IF :INPUT.VALUE < 5 [DO.COUNT
   :INPUT.VALUE + 1]
END
```

This procedure will print the values from 1 to 5 if you enter

```
DO.COUNT 1
```

Next, make a simple change as follows:

```
TO DO.COUNT :INPUT.VALUE
IF :INPUT.VALUE < 5 [DO.COUNT
   :INPUT.VALUE + 1]
PRINT :INPUT.VALUE
END
```

Now if you type

```
DO.COUNT 1
```

you see

```
5
4
3
2
1
```

What is happening? Examine a diagram at the bottom of this page that is similar to the one we looked at for COUNT.UP earlier

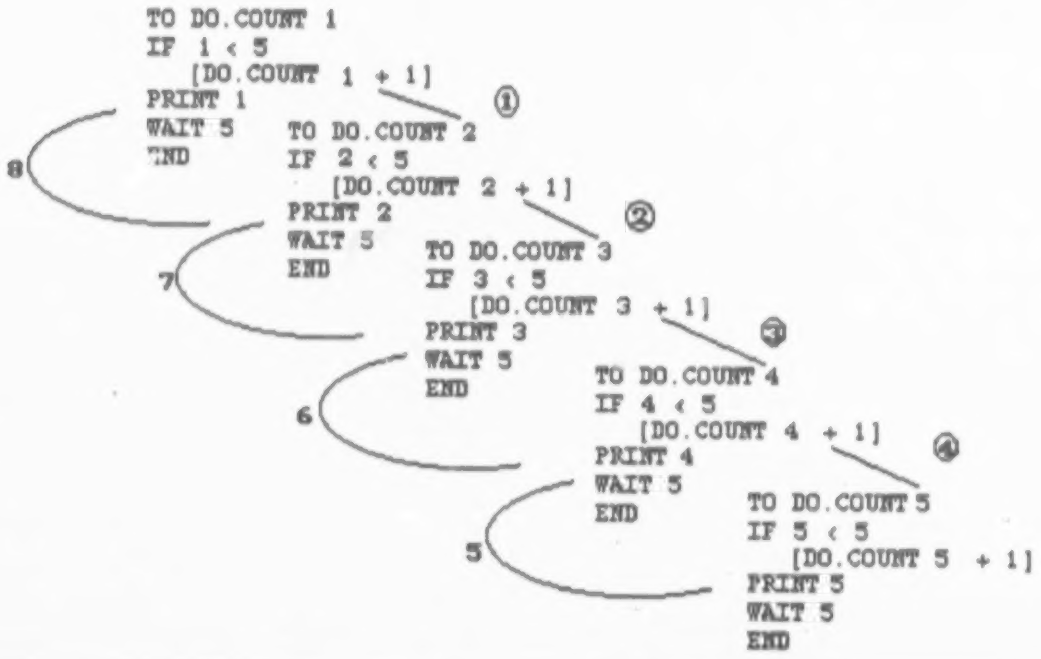
First, procedure DO.COUNT is called. The number 1 is clearly less than the number 5. Thus, at step (1) in the diagram, DO.COUNT is called with an input of 2. Notice that the PRINT 1 statement is not run at this time. Next 2 is tested against 5. The test fails, and DO.COUNT is called with an input value of 3 (at step (2) in the diagram) This process continues, until at step (4), 5 < 5 is false. Thus DO.COUNT is not called again, so the procedure moves to the next statement

```
PRINT 5
```

Thus, 5 is printed. Logo reaches the END of the procedure. Now, the procedure "above" is incomplete (step (5)). Logo finishes that procedure by running

```
PRINT 4
```

Now there remain the rest of the PRINT statements that were not run, so Logo runs those statements as it encounters them until it reaches an END statement, then it moves to the next unfinished procedure that was stacked up waiting to be run.



```
TO DO.COUNT 1
IF 1 < 5
   [DO.COUNT 1 + 1]
PRINT 1
WAIT 5
END
```

```
TO DO.COUNT 2
IF 2 < 5
   [DO.COUNT 2 + 1]
PRINT 2
WAIT 5
END
```

```
TO DO.COUNT 3
IF 3 < 5
   [DO.COUNT 3 + 1]
PRINT 3
WAIT 5
END
```

```
TO DO.COUNT 4
IF 4 < 5
   [DO.COUNT 4 + 1]
PRINT 4
WAIT 5
END
```

```
TO DO.COUNT 5
IF 5 < 5
   [DO.COUNT 5 + 1]
PRINT 5
WAIT 5
END
```

At first, embedded recursion seems quite mysterious. If you study the above example carefully, you will no doubt see why and how it works. However, embedded recursion is probably not something that you want to use just now in your programming. If you are working with other people learning Logo, you will often see the results of their unwittingly using embedded recursion. Embedded recursion is often the reason that a message flashes on the screen again and again when the author thought it should appear only once. Perhaps the screen flashes at the end of a program for unexplained reasons or there are unexpected pauses when a program should just stop. Such effects occur when statements such as CT or CG are placed after a recursive call.

When you have had more experience, you may want to explore this idea further. For a more complete treatment of recursion, see Brian Harvey's book *Computer Science Logo Style* mentioned in the introduction to this book.

TIPS AND TECHNIQUES

The procedures in the first part of this chapter become increasingly complex. If you are new to programming, you may well have trouble following the explanations. You can use the ideas you have learned about debugging to help you understand what is happening in a difficult procedure. For example, here is the last version of COUNT.UP with a couple of PRINT statements added to help you understand the procedure.

```
TO COUNT.UP :INPUT.VALUE :TO.ADD :MAX
PRINT :INPUT.VALUE
WAIT 5
(PRINT [MAX = ] :MAX)
(PRINT [TO.ADD = ] :TO.ADD)
IF :INPUT.VALUE < :MAX
   [COUNT.UP  (:INPUT.VALUE  +  :TO.ADD)
   :INPUT.VALUE :MAX]
END
```

You might then run this procedure by typing

```
COUNT.UP 5 3 15
```

and then you can study the step by step results. Then try changing each of the values to see what happens. Use this technique with any procedures that you have trouble understanding.

PROJECT SUGGESTIONS

The version of the COUNT.UP procedure on Page 94 does not actually print the maximum value, for example, COUNT.UP 3 5 9. See if you can fix it so the procedure runs correctly for any values you input.

Draw appropriate diagrams for the various versions of COUNT.UP. Can you explain the behavior of each of the modifications to COUNT.UP?

Experiment with other types of GROW procedures. Make triangular or hexagonal spirals. What happens if the spirals wrap? Can you modify the procedure so that the user can tell how big the spiral should get? How about modifying it so that the user can tell how large the space between sides of the spiral should be?

Modify the SEE.SHAPES procedure so that the number of the shape is printed under the shape.

Modify the SEE.SHAPES procedure so that the user can repeatedly specify which shape s/he wants to see. Hint: Add a question asking the user if s/he wants to see another shape.

Write a program containing DRAW.STARS and use RANDOM as input to it. Perhaps you could have the program stop and ask the user to count the stars. Then the program could check to see if the user counted correctly.

30. Logo Arithmetic and Math

Have you discovered that you can do math in Logo? For example, you can type

```
PRINT 3 + 4
```

and Logo prints

```
7
```

or

```
PRINT (3 * 4) - 10
```

and Logo prints 2.

Notice that the * is the sign that Logo uses for multiplication. Here are examples of the arithmetic and mathematics that Logo can do.

PRINT 20 + 5	Logo prints 25
PRINT 20 - 5	Logo prints 15
PRINT 20 * 5	Logo prints 100
PRINT 20 / 5	Logo prints 4
PRINT REMAINDER 20 5	Logo prints 0

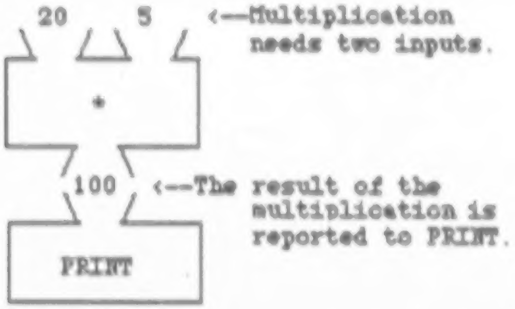
(The remainder when 20 is divided by 5)

If you have studied some trigonometry:

PRINT COS 20	Logo prints 0.9396
PRINT SIN 20	Logo prints 0.342.

(Note: this is 20 degrees)

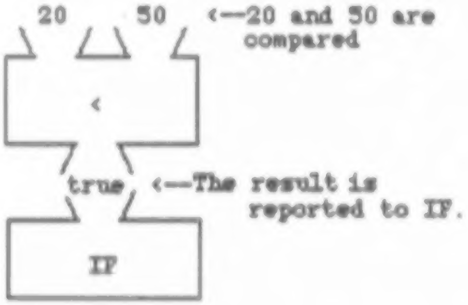
Notice all of these mathematics operations are reporters — they report a value to the PRINT command. For example, you can diagram PRINT 20 * 5 as follows:


20
5
<—Multiplication needs two inputs.
*
100
<—The result of the multiplication is reported to PRINT.
PRINT

In math, these are called binary operations because they have two inputs. SIN and COS are unary operations. They have one input. Can you diagram SIN 30?

You have used other arithmetic reporters with IF and IFELSE. The result of =, < or > was reported to IF.

IF 20 < 50 would be diagrammed as:



All three of these reporters return either true or false. When these are used in IF or IFELSE, the "true" or "false" is reported to the conditional. The conditional statement then selects an action based on the result reported ("true" or "false").

You can print the result of these reporters just as you can print the result of arithmetic reporters.

PRINT 20 = 5	Logo prints "false"
PRINT 20 < 5	Logo prints "false"
PRINT 20 > 5	Logo prints "true"

Using arithmetic reporters and what you know about incrementing, you can easily have Logo generate a table of values. For example

```
TO SQUARES :NUMBER :LARGEST
(PRINT :NUMBER  :NUMBER * :NUMBER)
IF :NUMBER < :LARGEST
   [SQUARES :NUMBER + 1 :LARGEST]
END
```

If you type

```
SQUARES 1 5
```

Logo prints

```
1   1
2   4
3   9
4   16
5   25
```

Using this idea, you can create multiplication tables or tables of sines easily.

With the skills you now have in Logo, you can write a simple arithmetic drill and practice program. Begin with a procedure to print a problem and accept the answer.

```
TO SEE.AND.CHECK :NUMBER1 :NUMBER2
PRINT []
PRINT []
(INSERT :NUMBER1 [+] :NUMBER2 [=])
IFELSE READWORD = (:NUMBER1 + :NUMBER2)
   [GOOD.JOB]
   [TRY.AGAIN :NUMBER1 :NUMBER2]
END
```

Examine the INSERT line carefully. Do you see why it prints the problem and not the answer? How is

```
    :NUMBER1 [+] :NUMBER2
```

different from

```
    :NUMBER1 + :NUMBER2
```

Do you see that in the first line the plus sign is quoted? In the second it is not. That means that the [+] will be printed, but when + appears without being quoted, it is treated as a reporter. It reports the sum of the two numbers. This is an important idea. Study the above example to be sure you see what is happening.

The procedures GOOD.JOB and TRY.AGAIN might look like this:

```
TO GOOD.JOB
CT
REPEAT 50 [INSERT [Good for you!!!!!! ]]
END

TO TRY.AGAIN :N1 :N2
PRINT []
PRINT [No, try the problem again]
SEE.AND.CHECK :N1 :N2
END
```

Notice that the numbers are sent to TRY.AGAIN, but not used. They are then sent back to SEE.AND.CHECK so that the same problem is printed again. That is, the value of N1 is sent back to SEE.AND.CHECK and becomes the value of NUMBER1 and the value of N2 is sent back to SEE.AND CHECK and becomes the value of NUMBER2. The procedures SEE.AND.CHECK and TRY.AGAIN make use of the idea of passing values from one procedure to another that you have seen earlier.

To get SEE.AND.CHECK to select random problems, you need a statement something like

```
SEE.AND.CHECK RANDOM 10 RANDOM 10
```

Here, the result from the first RANDOM 10 (a number from 0 to 9) becomes the value of NUMBER1. Similarly, a number from 0 to 9 is reported by the second RANDOM 10 to NUMBER2. Each number in the problem shown then ranges from 0 to 9. Then, a procedure is needed to allow the user to select how many problems s/he wants to practice.

```
TO ASK.USER
PRINT [How many problems do you want to do?]
REPEAT READWORD [SEE.AND.CHECK RANDOM 10
   RANDOM 10]
END
```

Now this program can be used to do arithmetic practice.

TIPS AND TECHNIQUES

You have seen that parentheses are often used in an IF or IFELSE statement when the reporters <, >, or = are used. If you do not use parentheses, sometimes mysterious bugs appear. This is because the standard mathematical symbols are "infix" notation. The symbol goes in between the inputs. That is, we write 3 + 4 or 5 < 6 rather than +3 4 or < 5 6. All Logo commands and reporters use "prefix" notation. That is, the inputs to the procedures or primitives come after the command or reporter. When infix and prefix notation are mixed, it can be difficult for the Logo parser to translate correctly. Parentheses force Logo to translate expressions in the way you intend.

PROJECT SUGGESTIONS

Write a program to generate a table of values that you might find useful. Format the page nicely, then print the results to use elsewhere.

Can you draw explanatory diagrams to explain how values are passed in the SEE.AND.CHECK program? Try explaining the program to someone who has never seen this idea before.

Modify the SEE.AND.CHECK program so that the user can give the range of numbers s/he wants.

LogoWriter has some other math reporters available. They are on a page called MATHTOOL. Get that page and experiment with the procedures found there.

31. OUTPUT

You have seen the OUTPUT command in the READWORD procedure that is used. Except for that procedure, all of the procedures that *you* have written have been commands. That is, when you typed the name of the procedure, it caused some effect — it did not report a value. OUTPUT is the command that allows you to write your own reporters. Procedures containing OUTPUT report values. Think about READWORD. If you type

```
READWORD
```

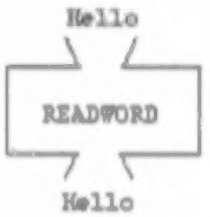
and then type

```
Hello
```

Logo responds

```
I don't know what to do with Hello
```

READWORD reported a value, but you did not tell Logo what you wanted to do with the value. You might diagram this as follows:


Hello
READWORD
Hello

Hello is reported but there is no command to "catch" it.

READWORD behaves just like other reporters that you have used, like RANDOM.

Suppose you wanted a procedure that reported the cube of a number. Remember that the cube of a number is that number times itself three times. For example

$$2^3 = 2 \times 2 \times 2 = 8$$

and

$$5^3 = 5 \times 5 \times 5 = 125$$

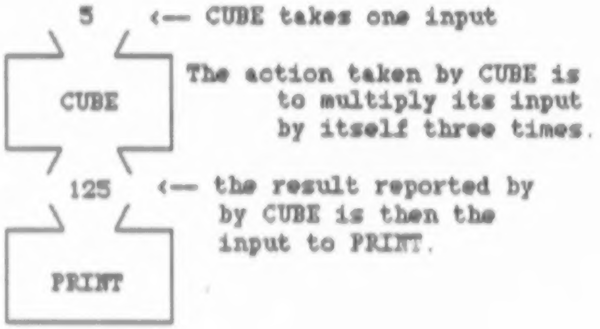
You can use OUTPUT to write such a procedure

```
TO CUBE :NUMBER
OUTPUT :NUMBER * :NUMBER * :NUMBER
END
```

Now, if you type

```
PRINT CUBE 5
```

Logo prints 125.


5
<— CUBE takes one input
CUBE
The action taken by CUBE is to multiply its input by itself three times.
125
<— the result reported by by CUBE is then the input to PRINT.
PRINT

Writing simple reporters can give you your own set of personalized tools to use in other procedures. For example, you can create a color "tool kit" using OUTPUT so that you don't have to remember the color numbers. You can write

```
TO ORANGE
OUTPUT 4
END
```

(The procedure assumes the color number for orange is 4.) If you then type

```
PRINT ORANGE
```

you see the number 4 on the page. Using PRINT with ORANGE lets you see the color number. However, there is another way to use ORANGE that makes your programs more readable. You can type

```
SETBG ORANGE
```

The background of the page turns orange. You can write a not have to remember color names, but other people reading your program can tell what colors you intended to use.

In a similar manner, you can write shapes tools:

```
TO CAT
OUTPUT 21
END
```

and then

```
SETSH CAT
```

to make the use of shapes clearer.

You can save groups of procedures, such as a color or shapes tool kit, on a separate page. When you want to use them on the page you are working on, you can type

```
LOAD "page.name.with.tool.procedures
```

These procedures are then *added* to the procedures currently on the page.

LogoWriter has two built in reporters that you have not seen before that give information about the color of the turtle and the color of the background. These can be used with a color tool kit to make procedures using them more readable. The first of these reporters is COLOR. If you type

```
PRINT COLOR
```

you see the number representing the current pen color. You can then put a statement such as

```
IF COLOR = ORANGE [SELECT.ANOTHER]
```

COLOR reports the current color number. ORANGE reports 4. If the current pen color is 4, then the procedure to select another color is called. See the illustration at the bottom of the page.

This might be used if you have a program to select random pen colors. Perhaps you have an orange background and you don't ever want to select an orange pen. You might write

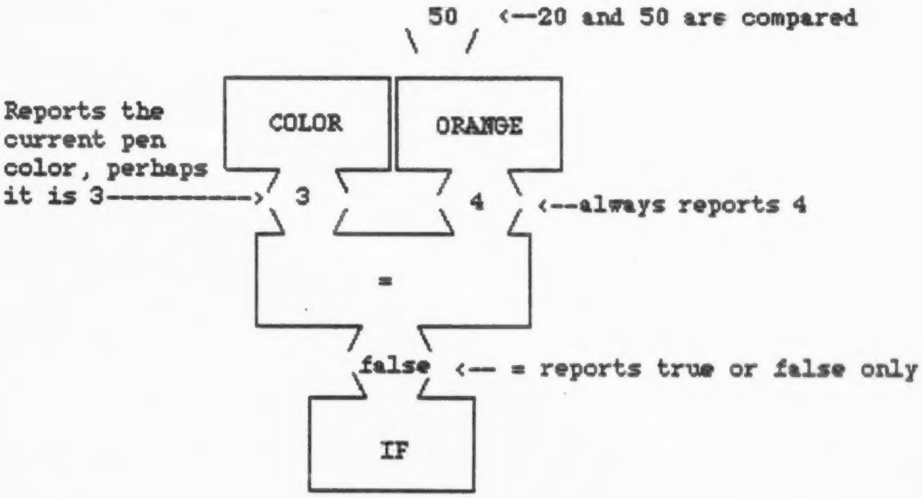
```
SETBG ORANGE

TO SELECT.ANOTHER
SETC RANDOM 6
IF COLOR = ORANGE [SELECT.ANOTHER]
END
```

Then your program could draw a design in a random color, which you then know will not be orange.

The other new color reporter is COLORUNDER. COLORUNDER reports the color under the turtle's pen. You might use

```
IF COLORUNDER = ORANGE [YOU.WIN]
```

in a game. Perhaps the goal is to land the turtle on an orange circle. This statement checks to see if the color underneath the turtle is orange. Note: if you have difficulties with COLORUNDER, you may need to move the turtle a few steps and try again. This has to do with the make up of the screen in some versions of LogoWriter.


50
<--20 and 50 are compared
Reports the current pen color, perhaps it is 3---------->
COLOR
ORANGE
3
4
<--always reports 4
=
false
<-- = reports true or false only
IF

In an earlier chapter, you learned to write music using LogoWriter. Writing music might be easier if you made a tool kit. For example, you can write

```
TO C4
OUTPUT 262
END

TO D4
OUTPUT 294
END

TO E4
OUTPUT 330
END
```

and so forth. Note that the numbers in the above procedures come from chart given in the Chapter 15 of this book.

You can even write procedures for quarter notes, whole notes, half notes, etc. So, for example

```
TO Q
OUTPUT 20
END

TO H
OUTPUT 40
END
```

Then you could write

```
TO PHRASE1
TONE E4 Q
TONE D4 Q
TONE C4 H
END

TO THREE.BLIND
REPEAT 2 [PHRASE1]
END
```

to play the first two phrases of Three Blind Mice in the 4th octave of the scale on the chart!

In the past if you wanted to write a procedure to calculate the area of a rectangle, you wrote:

```
TO AREA :LENGTH :WIDTH
PRINT :LENGTH * :WIDTH
END
```

Typing

```
AREA 8 10
```

causes Logo to respond

```
80
```

As long as you do not need to use the area elsewhere, this works fine. However, what if you want a procedure that makes use of the area, perhaps to find the cost of carpet? Then, if you write

```
TO AREA :LENGTH :WIDTH
OUTPUT :LENGTH * :WIDTH
END
```

you can use the value elsewhere. Thus, you can now type

```
PRINT 15 * AREA 8 10
```

to find the cost at $15 per square yard of an 8 x 10 carpet.

If you are comfortable with mathematical language, you can think of reporters as functions. That is, the AREA procedure above is a function that produces the area given the length and width. In Logo terms, it is a procedure which takes two inputs and reports the product of those inputs. These are just different ways of saying the same thing.

Writing procedures like this as reporters allows them to be used in a number of interesting ways. Suppose you want a procedure to report the perimeter of a rectangle.

```
TO PERIMETER :LENGTH :WIDTH
OUTPUT 2 * :LENGTH + 2 * :WIDTH
END
```

You can then use this procedure to make sure the perimeter of a rectangle is within a certain range.

```
TO CHECK.PERIMETER :L :W
IFELSE (PERIMETER :L :W) < 100
   [(PRINT :L [and] :W [are O.K.])]
   [(PRINT :L [and] :W [are too big.])]
END
```

This procedure can be used to plan a fenced-in area for which you had only 100 feet of fencing. Notice that the value of L is passed into LENGTH and that the value of W is passed into WIDTH in the PERIMETER procedure. PERIMETER then uses the values of LENGTH and WIDTH to produce the value that is reported back to CHECK.PERIMETER by PERIMETER. Can you draw diagrams to explain how the reporters function and how the values are passed from input to input?

Writing your own reporters can be useful both in adding tools to your programs and in making programs more readable. When you are writing a program and discover that you want a value computed and then used elsewhere in the program, you probably need to use OUTPUT.

It takes practice to learn to use OUTPUT. At first the idea of a command that turns a procedure into a reporter is a strange notion. Start by practicing with simple examples such as the tool kits at the beginning of this chapter. When you feel comfortable with them, move to more complex procedures that accept an input and report back an output, such as the CUBE procedure. In time you will feel more comfortable with OUTPUT. If you choose to learn more advanced programming in Logo, you will find that understanding OUTPUT is an essential part of mastering sophisticated list processing techniques.

TIPS AND TECHNIQUES

There are two ways in which OUTPUT is useful at this stage of your programming career. The first is the creation of tool kids as described in this chapter. These tool kits are made up of isolated procedures that are not connected with other procedures in your program. That is, procedures in a tool kit would not appear in the procedure tree for a program. Instead each procedure in the tool kit would have its own single-entry procedure tree.

The second use of OUTPUT is to create functions to produce some value. Writing procedures such as AREA above allows you to name a computation and then use the result of that computation anywhere in your program. In general, these functions will be a part of your procedure tree.

At this point in your Logo programming, you can think of OUTPUT as a method of naming actions or ideas in your program. You are giving a *name* to a color or shape number, or to a computation (such as AREA). Later in your Logo programming, you will discover that OUTPUT is the key to sophisticated programming ideas using Logo.

PROJECT SUGGESTIONS

Write your own math tool kit. Create procedures that you might use frequently and put them onto a page of your own tools. Be sure you put comments on the page so that it is clear what each procedure is for. (Use the MATHTOOLS page as a model.)

Use COLOR in a program that selects both the pen color and the background color randomly and draws designs but checks to be sure that the background color is not the same as the pen color. (Hint: Remember BG?)

Use COLORUNDER to create a simple "game." Perhaps have the turtle move randomly until it lands on a particular color. You could have a counter that prints how many moves the turtle makes until it lands on the right color.

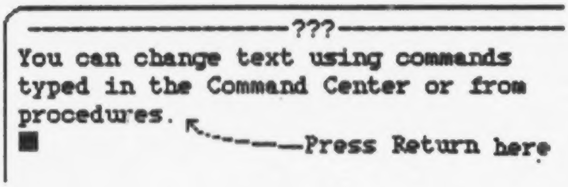
Make a music tool kit. You might use numbers to specify octaves. For example, you could have procedures called C1, C#1, D1,...C2, C#2, D2, etc. *Caution: If you create too many note procedures you may run out of memory. Write procedures for only a couple of octaves.*

Write a procedure to create music with random pitches and durations. How can you keep the pitches or durations above a particular number? (Hint: 5 + RANDOM 10 reports numbers from 5 to 14.)

32. Moving the Cursor From Procedures

You have used the word processor to create text on the page. You have also used the Select, Cut, Copy, and Paste keys to change both text on the page and procedures on the flip side of the page. All of the things that can be done using key combinations on the keyboard can be done from within a procedure. In this chapter you will begin to learn how to manipulate text from procedures.

Use the Up keys and type the text as shown below.

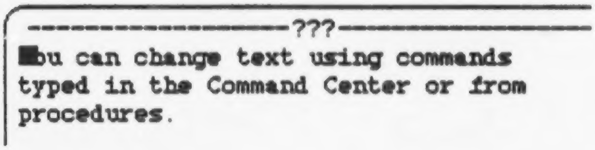


```
-------------------???-----------------
You can change text using commands
typed in the Command Center or from
procedures. <-------Press Return here
■
```

Now, use the Down keys to return to the Command Center. Type

```
TOP
```

You see

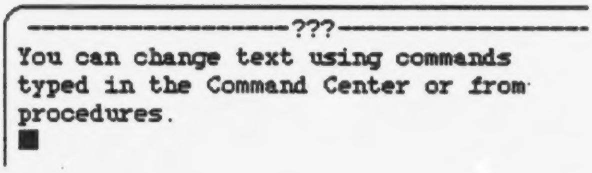


```
-------------------???-----------------
■ou can change text using commands
typed in the Command Center or from
procedures.
```

Next type

```
BOTTOM
```

You see

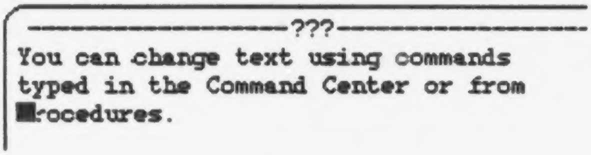


```
-------------------???-----------------
You can change text using commands
typed in the Command Center or from
procedures.
■
```

Next type

```
CU
```

The cursor moves up one line. You see

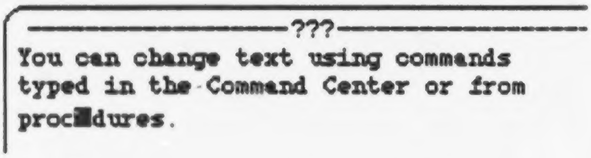


```
-------------------???-----------------
You can change text using commands
typed in the Command Center or from
■rocedures.
```

Next type

```
REPEAT 4 [CF]
```

The cursor moves forward 4 characters. You see

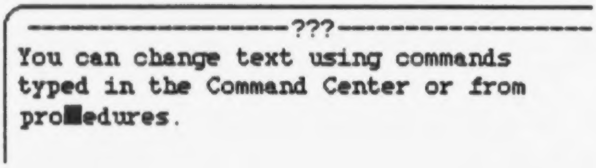


```
-------------------???-----------------
You can change text using commands
typed in the Command Center or from
proc■dures.
```

You can also move the cursor backwards by typing

```
CB
```

You see

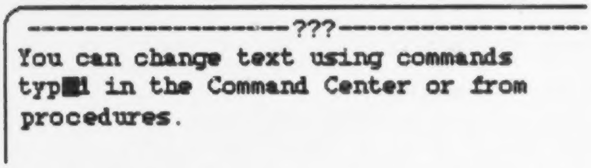


```
-------------------???-----------------
You can change text using commands
typed in the Command Center or from
pro■edures.
```

Next type

```
CU
CU
CD
```

The cursor moves up two lines and down one line. You see



```
-------------------???-------------------
You can change text using commands
typ█d in the Command Center or from
procedures.
```

The cursor moves up two lines and down one line. You see

Next, try

```
TOP
EOL
```

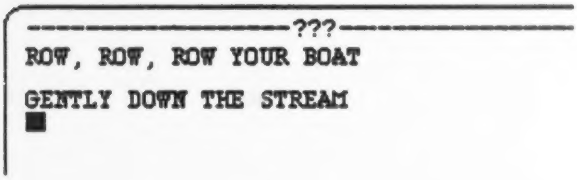
The cursor moves to the end of the *paragraph*, as if you pressed next screen. Type

```
SOL
```

The cursor moves to the start of the paragraph. Keep in mind that "end of line" means move to the next Return character, and start of line means move to the previous Return character.

These commands can be used to add interesting effects to your procedures. Suppose you put the lines of a song on the page. For example, you might have

```
TO ROW.ROW
PRINT [ROW, ROW, ROW YOUR BOAT]
PRINT []
PRINT [GENTLY DOWN THE STREAM]
PRINT []
```



```
-------------------???-------------------
ROW, ROW, ROW YOUR BOAT

GENTLY DOWN THE STREAM
█
```

```
END
```

Perhaps you would like to have the cursor move to each line and then play the music that goes with these words. So, you might have

```
TO SEE.AND.SING
ROW.ROW
TOP        <— Moves cursor to the first line, on the word ROW.
PLAY.PHRASE1   <— A procedure to play the first phrase.
CD       <—These two commands move the cursor
CD       to the word GENTLY.
PLAY.PHRASE2    <— A procedure to play the second phrase.
END
```

With the ability to move the cursor to the top and bottom of your text as well as one paragraph, one line or one character, you can place the cursor wherever you wish after you put text on the page.

TIPS AND TECHNIQUES

Each time you press Return/Enter when using the LogoWriter word processor or use a PRINT statement to put text on the page, there is an invisible "Return" character at the end of each line. As you are working with these new commands, pay attention to the position of the cursor as it moves. If you type

```
PRINT [HI]
```

and then type

```
TOP
BOTTOM
```

you will see the cursor placed on the Return character at the beginning of the line below the word HI.

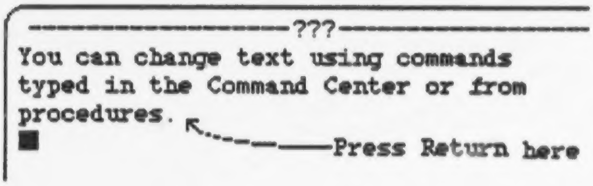
PROJECT SUGGESTIONS

Write a brief poem on the screen. Have the cursor move to each word, in the rhythm that you would read the poem.

Choose a different song than Row, Row, Row Your Boat. Put the lyrics on the page and have the cursor move to the appropriate words as the song is played.

33. Using Editing Commands From Procedures

You have learned to use Select, Cut, Copy, and Paste to edit text in the word processor. There are LogoWriter commands which allow you to perform these same functions from procedures. Suppose you have the following text on the page. Now, explore how the commands SELECT, COPY, and PASTE work. Type



```
-----------------???-----------------
You can change text using commands
typed in the Command Center or from
procedures.
■
```

```
TOP
SELECT
```

The cursor disappears. Then enter

```
REPEAT 3 [CF]
```

The word "You" is then highlighted. Next type

```
COPY
```

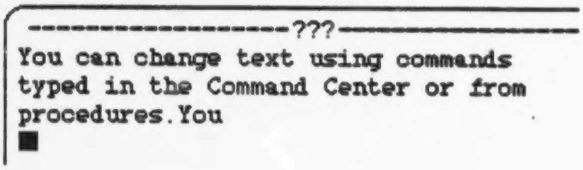
The word "You" is no longer highlighted. Then type

```
BOTTOM
CB
```

and the cursor moves to the end of the text. Finally, type

```
PASTE
```

and the paragraph looks like this:

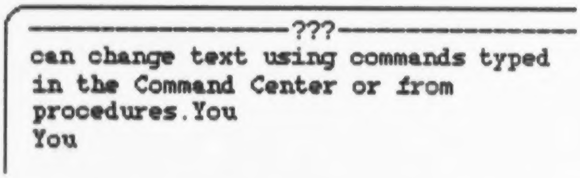


```
-----------------???-----------------
You can change text using commands
typed in the Command Center or from
procedures.You
■
```

You have not yet used CUT. Try typing

```
TOP
SELECT
REPEAT 4 [CF]
CUT
BOTTOM
PASTE
```

and the paragraph looks like this:



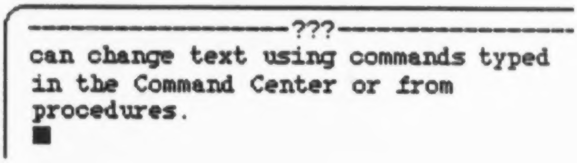
```
-----------------???-----------------
can change text using commands typed
in the Command Center or from
procedures.You
You
```

The word "You" was selected and then cut from the beginning of the sentence and pasted at the end of the paragraph at the position of the cursor. The text automatically adjusted on the page just as it does when you are using the word processor.

There is also a command that corresponds to the Delete or Backspace key. Using the text that is already on the page, type

```
REPEAT 5 [CB]
REPEAT 6 [DELETE]
```

and the paragraph looks like this:



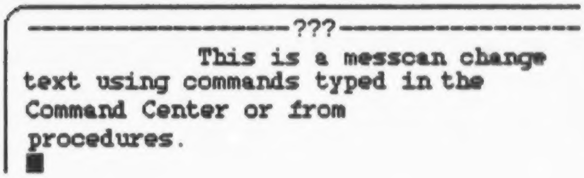
```
-----------------???-----------------
can change text using commands typed
in the Command Center or from
procedures.
■
```

Note: The DELETE command deletes the character under the cursor. The backspace and Delete keys work somewhat differently on different kinds of computers. Take a moment to compare the way the command DELETE works with your Delete/Backspace key.

There is also a command that corresponds to the TAB key. Try typing

```
TOP
TAB
TAB
INSERT [This is a mess]
```

The cursor moves to the top and moves forward to column 5 after the first TAB command and then moves to column 10 after the second TAB command. The text moves out of the way. The phrase "This is a mess" is then added to the paragraph:



```
------------------???------------------
             This is a messcan change
text using commands typed in the
Command Center or from
procedures.
█
```

Here are a couple of examples using the commands that you have just learned. Suppose you want to write a procedure that would delete the first 5 characters from a sentence on the page. You might write

```
TO DELETE.5
TOP
REPEAT 5 [DELETE]
END
```

or you could write

```
TO DELETE.5
TOP
SELECT
REPEAT 5 [CF]
CUT
END
```

Suppose you wanted to put a title on a paragraph that is already on the page. You could write

```
TO TITLE
TOP
PRINT [This is my title]
PRINT []
END
```

and a title appears.

Perhaps you want to put a last line on a page.

```
TO END.IT
BOTTOM
PRINT []
PRINT [This is really the end!]
END
```

Anything you can do with keys, you can do from a procedure. When you have learned a bit more about text processing commands and reporters, you will find that you can do some useful and interesting projects. For now, concentrate on understanding how each of the new commands works.

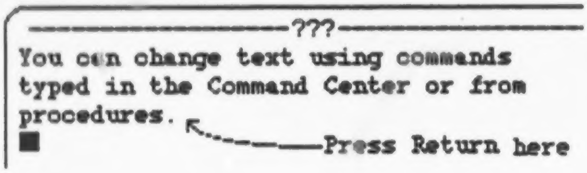
PROJECT SUGGESTION

Spend some time practicing using these new commands. Type a paragraph on the page and then manipulate it from the Command Center. When you are comfortable with the commands, try writing some procedures to move groups of letters from one place to another on the page.

34. More on Text Manipulation

You have learned how to use commands from the Command Center or procedures to move around in and modify text on the page. All of the commands that you learned did the same thing as keys on the keyboard. There are other primitives for manipulating text, some of which don't correspond to key strokes. These commands and reporters give you even more capabilities for working with text.

Again, start with this text on the page,



```
-------------------???-------------------
You can change text using commands
typed in the Command Center or from
procedures. ↖
■           `-------Press Return here
```

and then type

```
TOP
SEARCH "commands
```

The word "commands" is highlighted. SEARCH is a command that allows you to locate any word you want in the text on the page.

Next, try typing

```
SHOW SELECTED
```

Logo prints

```
commands
```

in the Command Center. You can type

```
UNSELECT
```

to remove the highlighting. Next, try typing

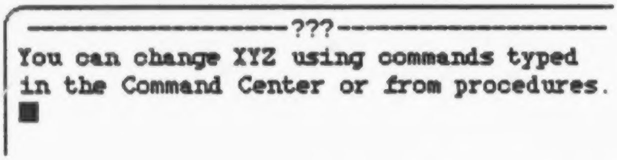
```
TOP
CD
SELECT
EOL
SHOW SELECTED
```

This time the entire paragraph is highlighted. Logo prints the last part of the paragraph. SELECTED is a reporter that returns the most recent text that is searched for or selected. (Remember to use UNSELECT to get rid of the highlighting.)

Next, type

```
TOP
REPLACE "text "XYZ
```

The cursor moves to the word "text," deletes it, and puts XYZ in its place. REPLACE is a command that allows you to replace a single word with any other word that you want.

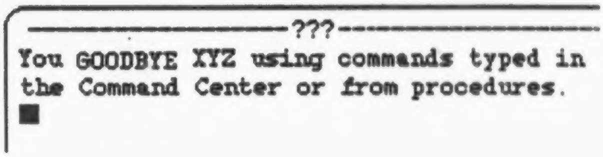


```
-------------------???-------------------
You can change XYZ using commands typed
in the Command Center or from procedures.
■
```

You can replace more than one word by typing

```
TOP
REPLACE "can\ change "GOODBYE
```

The backslash causes LogoWriter to view "can change" as one word. It then deletes both "can" and "change" and replaces them with the word "GOODBYE."



```
-------------------???-------------------
You GOODBYE XYZ using commands typed in
the Command Center or from procedures.
■
```

You can also use the backslash with SEARCH to search for more than one word. Type

```
TOP
SELECT
REPEAT 12 [CF]
COPY
SHOW CLIPBOARD
```

Logowriter prints the selected characters.

```
You GOODBYE
```

CLIPBOARD is a reporter that returns the most recent text that has been copied or cut. That means that when you cut a section of text out, you can get it back later by typing

```
PRINT CLIPBOARD.
```

at the appropriate spot on the page.

There is a difference between SELECTED and CLIPBOARD. Carefully examine the following sequence of commands. With the most recent text on the page, type

```
TOP
SELECT
REPEAT 12 [CF]
SHOW SELECTED
UNSELECT
```

You see

```
You GOODBYE
```

in the Command Center. Logo printed the text that was just highlighted.

Now type

```
TOP
SELECT
REPEAT 12 [CF]
COPY
SHOW CLIPBOARD
```

You also see

```
You GOODBYE
```

Now type

```
TOP
SELECT
CD
REPEAT 3 [CF]
SHOW SELECTED
UNSELECT
```

As expected, the word "the" appears in the Command Center. Now, type

```
SHOW CLIPBOARD
```

You see

```
You GOODBYE
```

This time, CLIPBOARD and SELECTED reported different things. SELECTED reports the currently selected text; CLIPBOARD reports the most recent text that was cut or copied.

These new commands give you flexibility both in writing procedures and in working with text in the word processor mode. For example, suppose you typed a paragraph and discovered that you misspelled the word "could." Perhaps you spelled it "culd." You can go to the Command Center and type

```
TOP
REPLACE "culd "could
```

Using the up arrow key and pressing Return repeatedly allows you to replace *all* of the misspellings at once. You could even write a procedure to accomplish this task.

```
TO FIX
TOP
FIX.ALL
END

TO FIX.ALL
REPLACE "culd "could
IF FOUND? [FIX.ALL]
END
```

FOUND? is a new reporter. It returns "true" if REPLACE (or SEARCH) finds its target, "false" otherwise. In fact, you can write a procedure to replace any word you want:

```
TO FIX
TOP
FIND.WHAT
END

TO FIND.WHAT
TYPE [Replace what?]      <—This pair of statements gets
SHOW "                      rid of the [ ]'s that SHOW
WITH.WHAT READWORDCC        produces.
END

TO WITH.WHAT :OLD.WORD    <—This procedure "holds"
TYPE [With what?]           the word to bereplaced.
SHOW "
FIX.ALL :OLD.WORD READWORDCC    <—Remember
END                   that you have to define READWORDCC.

TO FIX.ALL :OLD :NEW
REPLACE :OLD :NEW
IF FOUND? [FIX.ALL :OLD :NEW]
END
```

These new commands give you the ability to write procedures to all sorts of useful procedures to use with text on the page. These could include both procedures to help with word processing and procedures to modify the text on the page.

Recall the program from an earlier chapter to move the cursor to the beginning of each line of Row, Row, Row Your Boat? With SEARCH, you can highlight each word as the appropriate notes are played.

Suppose you have the line

```
ROW, ROW, ROW, YOUR BOAT
```

on the page. You can write the following procedure to play the music while highlighting the words.

```
TO ROW.ROW
TOP
REPEAT 2 [SEARCH "ROW TONE 262 20]
SEARCH "ROW
TONE 262 15
SEARCH "YOUR
TONE 294 5
SEARCH "BOAT
TONE 330 20
END
```

When you are working with text, you sometimes need to know whether you are at the end of the text on the page. The reporter TEXTLEN returns the length of the text on the page. Thus, if you type

```
CT
PRINT [HELLO]
SHOW TEXTLEN
```

Logo prints

```
6
```

This corresponds to the number of letters in "HELLO" plus the carriage return character. TEXTLEN always returns a number corresponding to the number of characters of the text on the page.

Sometimes you may also need to know where the cursor is on the page. The reporter TEXTPOS returns the current location of the cursor. Thus if you type

```
SHOW TEXTPOS
```

Logo prints the character number of the current cursor position. The number reported by TEXTPOS is counted from the beginning to the text on the page.

Not only can you find out the current location of the cursor. You can also place the cursor wherever you wish. Thus if you type

```
SETTEXTPOS 30
```

The cursor moves to the 30th character from the beginning of the text on the page.

Using TEXTLEN and SETTEXTPOS, you can have a "dancing cursor." Since TEXTLEN gives you the number of characters of text, you can select a random number corresponding to a cursor position within the text on the page using

```
1 + RANDOM TEXTLEN
```

This line will report a number between 1 and the length of the text. Now, putting this into a procedure using SETTEXTPOS,

```
TO CURSOR.DANCE
REPEAT 20 [SETTEXTPOS 1 + RANDOM TEXTLEN]
END
```

Try this procedure on some text of your choice. This procedure is a fun way to create an interesting effect.

There are two other primitives that make a "connection" between the turtle and text. If you type

```
CT
PRINT [HELLO]
SHOW CURSORPOS
```

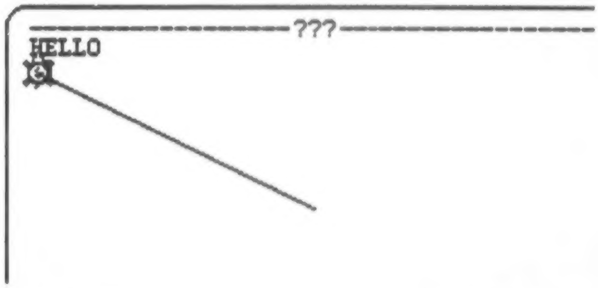
you see

```
[-137 76]
```

This is the location of the cursor in turtle coordinates. If you now type

```
SETPOS CURSORPOS
```

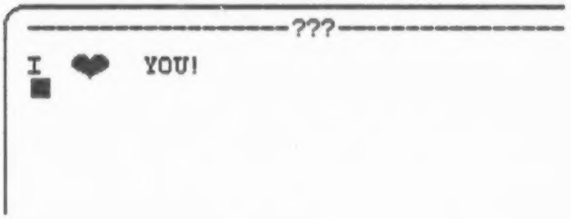
you see



The turtle moves on top of the cursor. This interaction makes the creation of a rebus from a procedure quite simple. For example, you might type

```
CG
CT
PRINT []
PRINT [I LOVE YOU!]
TOP
SEARCH "LOVE
CUT                        Recall that CHAR 32 represents
REPEAT 5 [INSERT CHAR 32]  <— a blank space.
REPEAT 3 [CB]       <—Moves the cursor to the middle of
SETSH 14                   the space.
PU
SETPOS CURSORPOS
BOTTOM              <—This gets the cursor out from under
PD                         the turtle.
STAMP
```

You then see



```
-----------------???---------------
I ❤  YOU!
■
```

This interaction between graphics and text can lead to a lot of interesting projects.

TIPS AND TECHNIQUES

Although you can see the text being manipulated when you write procedures to manipulate text on the page, sometimes the action occurs so rapidly that it is difficult to tell what is happening. Try inserting a WAIT command after each action, such as a cursor move or a REPLACE.

Text manipulation procedures are an excellent place to use the idea of inserting diagnostic statements. However, you will want to use SHOW, not PRINT since a PRINT statement will insert text on the page you are manipulating. Thus you might insert such statements as

```
SHOW TEXTLEN
SHOW TEXTPOS
SHOW CURSORPOS
```

to help you keep track of the action in your procedure.

Writing procedures that repeat some action until you reach the end of the text on the page can be a bit tricky. For example, suppose that you write

```
TO FIND.A
SEARCH "A
IF NOT (TEXTLEN=TEXTPOS) [FIND.A]
END
```

This procedure is supposed to search for the letter A until the end of the text is reached. In fact, this procedure will not work as intended. When the last letter "A" is found, the cursor simply stays on the "A" and blinks. The end of the text is never reached. This is simply because of the way the SEARCH command works. One way to fix this procedure is to write

```
TO FIND.A
SEARCH "A
IF FOUND? [FIND.A]
END
```

Now the procedure continues to search until no more "A's" are found. In general, when the cursor stops before the end of a text being searched, it has to do with the condition after IF. If you have such bugs, try a different stopping condition.

PROJECT SUGGESTIONS

Do you have some "favorite" spelling errors? Write a procedure to check for at least one of your errors and correct it if it is found.

When you are typing, do you forget to put two spaces after punctuation marks at the ends of sentences? Write a procedure to check for ". " (period blank) and replace it with ". " (period blank blank).

Write a procedure to encode text on the page. You might replace spaces with a letter, put numbers between every other letter, or add letters after certain vowels. When you get your encode procedure to work correctly, write a decode procedure to put it back in readable form.

Write a procedure that will find all of the occurrences of a word and replace it with a stamped shape. For example, you might find the word "cat" every time it appears and replace it with the cat shape.

With the commands you now know, you can write a procedure to write a procedure. All you need are PRINT and FLIP. Can you extend this idea and write a program that accepts commands from the keyboard and puts them into a procedure on the flip side of the page?

35. Using MAKE

In much that is written about Logo, you see the primitive MAKE used. Program examples in books and magazines use MAKE frequently. In fact you may have been wondering why this book has not introduced MAKE before.

If you know another programming language, you may have been wondering why this book hasn't taught you how to assign values of variables. Indeed the MAKE statement in Logo is used for the same purpose as LET in BASIC and in the same way as the assignment operator (:=) in Pascal. MAKE is used to assign values to names.

So, why has MAKE not been introduced earlier? The primary reason is that in Logo MAKE is frequently over used. It is often unnecessary and simply clutters both the code and the Logo system's memory. Learning to pass values among procedures is a necessary skill to learn to avoid over using MAKE. Using MAKE sparingly will result in better programming style and will help you as you move into more advanced programming in Logo.

There *are* places where using MAKE is all but necessary and where its use is appropriate. For example, it is not uncommon to want to increment (add to) or decrement (subtract from) a value at a number of points within a Logo program. While procedure inputs can sometimes be used to accomplish this task, there are other instances where the use of MAKE is more sensible. Sometimes you want to collect information that is to be used throughout a Logo program. This might be accomplished with DATA statement in BASIC or with sequential files in Pascal. Using MAKE is sometimes the solution in Logo.

In this chapter, you will learn the relationship between names created by MAKE, and names created as procedure inputs. You will also learn how to work with names created using MAKE.

You have used procedure inputs to associate a name with a value. For example,

```
TO GREET :WHOM
(PRINT [Hi,] :WHOM
END
```

If you type

```
GREET "Mike
```

Logo responds

```
Hi, Mike
```

Within the procedure, you use :'s (dots) to see the value of the name of a procedure input.

What happens if you go to the Command Center and type

```
SHOW :WHOM
```

Logo says WHOM has no value. Why? When you typed

```
GREET "Mike
```

"WHOM" had the value "Mike" within the procedure.

The explanation lies in the fact that procedure inputs are local to the procedure in which they are defined. That is both the name of the procedure input (in this case WHOM) and the value associated with the name (in this case Mike) cease to exist when the procedure finished running.

> Technical note: Procedure inputs are actually defined as long as the procedure is active. So, if a procedure calls another procedure, the name is actually accessible in that subprocedure. This is called **dynamic scoping.** If you use dynamic scoping to access the name of a variable in a subprocedure, it destroys your ability to test procedures independently of other procedures. Thus it is best not to rely on it.

There are times, however, that it would be convenient to have a name retain its value throughout a program. One place it is convenient to have a value available throughout a program is when you want to use the name of the user several times in a program. To do this you can use a procedure like this:

```
TO ASK.NAME
PRINT [What is your name?]
MAKE "USER READLIST
END
```

Notice the " mark in front of USER. You use the " mark to define a name. A Logo name is a word and you will remember that the quotation mark is used to quote a name. Then, anywhere in the program, you can use

```
PRINT :USER
```

and the value assigned to USER will be printed. We say that this name is **global** to indicate that it is understood everywhere in the program.

Technical note: In LogoWriter, global names retain their value from page to page. Thus, you can use MAKE as one way to transfer information from one page to another.

The general form of the MAKE command is

```
MAKE some name some value
```

You can read this as follows: "MAKE *some name* have the value of *some value*." So, for example, if you have

```
MAKE "THE.NAMES [MARY SAM TOM SUE]
```

you read "MAKE the name THE.NAMES have the value of the list MARY, SAM, TOM, and SUE. In this case, THE.NAMES is a Logo name and must be quoted to be used with MAKE since the first input to MAKE must be a Logo word.

Since global names become a permanent part of the page, some commands are needed to examine and manipulate them. In order to see what names are present, you can type

```
PRINTNAMES
```

to display all of the names (and their values) on the page, or

```
SHOWNAMES
```

to display all of the names (and their values) in the Command Center.

If you wish to get rid of all the names on the page, you can type

```
CLEARNAMES
```

If you wish to get rid of a name, you can type

```
CLEARNAME "USER
```

or, if there are several that you want to remove, you can type

```
CLEARNAME [USER OLD  NAME  WHICH.SPOT]
```

With these tools, you can identify and work with global names.

At this point you may have decided that MAKE always creates global names. Unfortunately, things are not that simple. You have seen that procedure inputs create local names, for example in the procedure GREET.

```
TO GREET :WHOM
(PRINT [Hi,] :WHOM
END
```

You also saw how MAKE could create a global name.

```
TO ASK.NAME
PRINT [What is your name?]
MAKE "USER READLIST
END
```

Consider this example.

```
TO CHANGE :NUMBER
PRINT :NUMBER
MAKE "NUMBER :NUMBER + 1
PRINT :NUMBER
END
```

If you run this procedure by typing

```
CHANGE 23
```

you see

```
23
24
```

on the page. Is NUMBER local to the procedure CHANGE? Try typing

```
PRINT :NUMBER
```

or

```
PRINTNAMES
```

You see that NAME is not a global name. It has no value outside of CHANGE. Using MAKE to change the value of a local name does not make it global.

Here are some "rules of thumb" to help you better understand how names work.

- When MAKE is used outside of a procedure, it always creates a global name.
- When MAKE is used inside of a procedure to create a new name, that name is global.
- When MAKE is used to change the value of a procedure input (local name) that name remains local.

You should take some time to internalize these "rules" so that when you encounter MAKE, you can think about it correctly.

TIPS AND TECHNIQUES

In spite of the discussion at the beginning of this chapter about not over using MAKE, there are places where it is entirely appropriate to make use of the ability to assign global values to names. Many people, as they move into more advanced Logo, find using a series of MAKE statements helpful in understanding complex Logo instructions. A simple example follows.

```
MAKE "CHOICE RANDOM 31
MAKE "SHAPE.NUMBER 1 + :CHOICE
SETSH :SHAPE.NUMBER
```

Before going any further, take a moment to examine the punctuation used with MAKE. You should read the second MAKE statement as follows: "Make the name SHAPE.NUMBER have the value of 1 plus the value associated with the name CHOICE". Using MAKE is one place where most people have some difficulty with punctuation. Remember that " quotes a word and : means the value associated with a name. The first input to MAKE is a word and the second is any Logo object.

Notice that the three statements given above can be collapsed as follows:

```
MAKE "SHAPE.NUMBER 1 + RANDOM 31
SETSH :SHAPE.NUMBER
```

since the value of CHOICE is RANDOM 31. Or, you can just write

```
SETSH 1 + RANDOM 31
```

since the value of SHAPE.NUMBER is 1 + RANDOM 31.

As this last command illustrates, neither the name SHAPE.NUMBER, nor the name CHOICE is really needed to accomplish the task, but using several global names may be helpful in developing understanding of the problem to be solved in writing a particular procedure. Feel free to use the MAKE statement in your work to help you learn a new idea, especially in the following chapters on list processing. However, when you complete a project, you should attempt to "collapse" your multiple MAKE statements as much as possible. Your final work should contain only necessary (or reasonable) uses of MAKE. As your understanding of Logo programming grows, you will find this task increasingly easy.

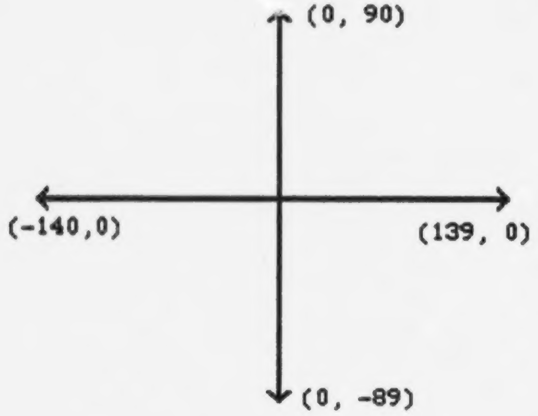
PROJECT SUGGESTION

Experiment with MAKE. Be sure that you see how it works. Have you tried all of the statements in this chapter? If not, do so. Write some short procedures using MAKE. Compare the results with using procedure inputs.

36. Cartesian Commands

When you work with the turtle in the "normal" turtle graphics, you use "body centered" commands. That is, the turtle moves relative to its current position. If you say RIGHT 90, the turtle turns right from where it currently is positioned. Similarly, if you say FORWARD 50, the turtle moves forward from its current position without changing direction.

Sometimes it is useful to be able to move the turtle to a particular position on the page from within a procedure, much as turtle cursor mode allows you to position the turtle on the page. It is also useful to be able to have the turtle face a particular direction. The commands that do this are based on graphing, similar to that which you may have done in math class. You can think of the page as a sheet of graph paper. For example, if you are using an Apple computer with LogoWriter, the "graph paper" looks like this:

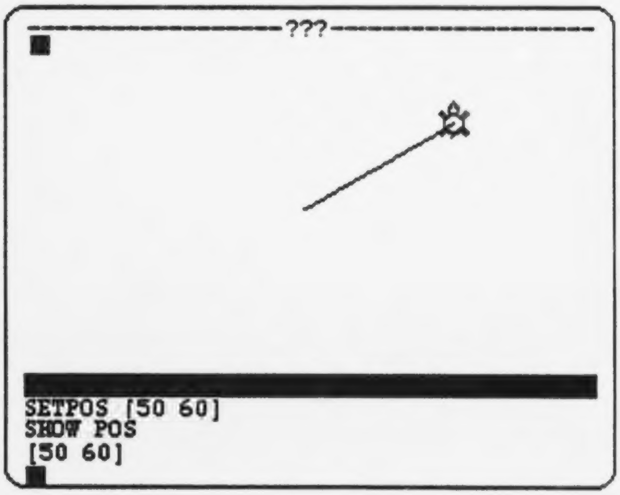


Notice that the origin, the point (0, 0) is in the middle of the page, where turtle 0 is found when you start LogoWriter.

To place the turtle at a particular position on the screen, you can type

```
SETPOS [50 60]
```

The turtle moves 50 steps to the right of the center of the page and 60 steps up. If the pen is down, the turtle leaves a trail.



```
SETPOS [50 60]
SHOW POS
[50 60]
```

You can find the location of the turtle by typing

```
PRINT POS
```

and the x- and y-coordinates of the turtle's position are printed.

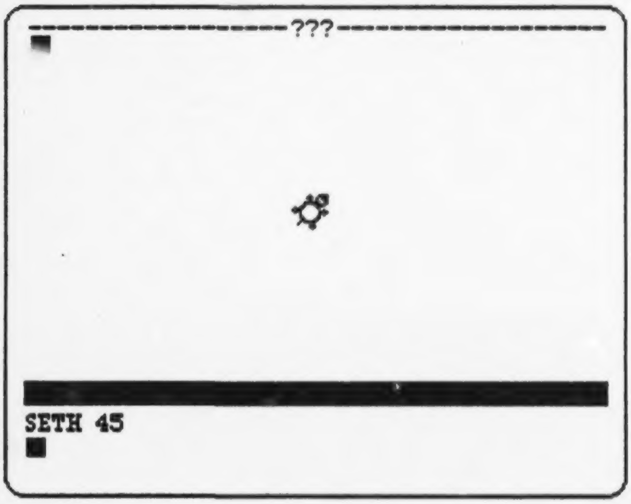
You can use SETPOS to create a point graphing program. For example, you might type

```
TO GRAPH.POINT.
SHOW [What point do you want to graph?]
PU
SETPOS READLISTCC
PD
FORWARD 1
SHOW [Another?]
IF READCHAR = "Y [GRAPH.POINT]
END
```

You can also specify the direction that the turtle faces. Type

```
SETH 45
```

and the turtle turns to face the upper right hand corner of the page.

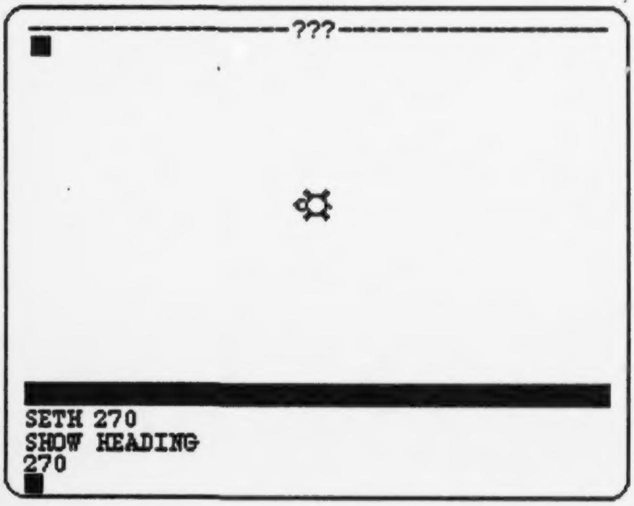


```
???
SETH 45
```

Type

```
SETH 270
```

and the turtle faces the left edge of the page.



```
???
SETH 270
SHOW HEADING
270
```

You can also check the direction the turtle is heading by typing

```
SHOW HEADING
```

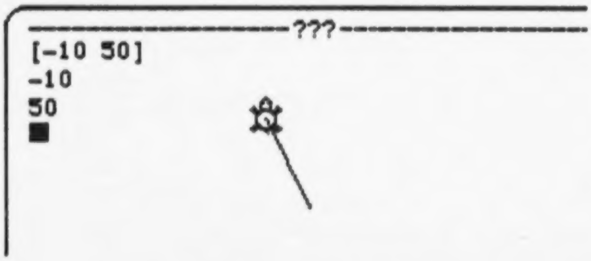
The current direction in which the turtle is facing appears in the Command Center.

Cartesian commands and reporters can be useful in a variety of types of programs. Suppose you are writing a game program and you want to know the location of the turtle. You can check the position of the turtle to see where it is on the page. You saw that POS reports both the x- and y- coordinates of the turtle position. The primitive XCOR reports the x-coordinate of the turtle, while YCOR reports the y-coordinate of the turtle. (Note: If you are using LogoWriter Version 1.0 or 1.1, then substitute FIRST POS for XCOR and LAST POS for YCOR.)

So, for example, if you type

```
SETPOS [-40 50]
PRINT POS
PRINT YCOR
PRINT XCOR
```

you see



```
???
[-10 50]
-10
50
```

Some newer versions of Logowriter contain a DISTANCE reporter. DISTANCE gives the number of turtle steps from a point to the current turtle location.

TIPS AND TECHNIQUES

Earlier in this book you saw that some of these Cartesian reporters could be helpful in your work with "standard" turtle graphics. For example, you could place the turtle with Turtle Move and then use

```
SHOW POS
```

to determine its location so that you could use SETPOS in a procedure. Often these Cartesian reporters can be very helpful in debugging programs. Putting statements such as

```
SHOW POS
SHOW HEADING
```

in a procedure in which the turtle is moving gives you a running account of what is happening. This can be very helpful in an animation program.

In general, reporters that return some information about the turtle or the cursor are good tools to use when trying to determine exactly what your program is doing. Keep in mind that statements such as

```
SHOW CURSORPOS
SHOW TEXTPOS
SHOW COLORUNDER
SHOW COLOR
SHOW BG
SHOW CLIPBOARD
SHOW SHAPE
```

are valuable tools that let Logo help you debug your programs.

PROJECT SUGGESTIONS

Practice using these coordinate commands. Be sure that you understand how each works. Write some procedures using coordinate commands to draw shapes.

Complete the graphing program started above. Add labels to the axes. Put "tic marks" on the axes and label them as well.

Create a game program. Decide on a target. Test to see if the turtle has landed on the target.

37. Predicates

The word "predicate" is a new term used to categorize procedures. It refers to a special *kind* of reporters. You have seen and used quite a few reporters. Different reporters report different kinds of Logo objects. For example, RANDOM reports numbers while CLIPBOARD returns words. The term "predicate" refers to reporters that return *only* true or false.

Why do you need such a class of reporters? Think about what goes after IF or IFELSE. You have always used something like =, < or >. Earlier you saw that these symbols are actually reporters. Since they return only true or false, they belong to the special class of reporters called predicates. For example

```
SHOW  3 = 4
```

displays

```
false
```

while

```
SHOW 3 < 4
```

displays

```
true
```

Most of the new reporters which you will learn about in this chapter end in a "?". (In many versions of Logo, predicates end in "p", for predicate.) There is nothing magic about the "?" at the end. It is just there to alert you that you that you are working with a predicate. Putting a "?" at the end of a procedure name does NOT turn that procedure into a predicate. Keep in mind that what makes a procedure a predicate is the fact that it reports *only* true or false.

You have also seen FOUND? Recall that, FOUND? returns true if a word is located and false otherwise. If you type

```
SEARCH "The
```

and if "The" is in the text being searched, then you can type

```
SHOW FOUND?
```

and

```
true
```

appears in the Command Center.

There are two reporters that are closely related to =. The reporter EQUAL? behaves the same way as the equal sign except that both inputs must *follow* EQUAL?. If you type

```
SHOW "A = "a
```

Logo displays true. If you type

```
SHOW EQUAL? "A "a
```

then Logo also displays true. The EQUAL? reporter does not distinguish between capital and small letters. Try typing

```
SHOW IDENTICAL? "A "a
```

Logo displays false in the Command Center. IDENTICAL? can be used to check for capitalization.

There are three predicates that are used with IF and IFELSE to create compound conditions. These reporters are NOT, AND, and OR.

NOT "reverses" the output from whatever follows it.

```
SHOW 3 = 4
false
SHOW NOT 3 = 4
true
```

AND and OR are used when you want to examine more than one condition.

```
SHOW 5 > 0
true
SHOW 5 < 10
true
SHOW AND (5 > 0) (5 < 10)
true

SHOW 5 > 0
true
SHOW 5 > 10
false
SHOW OR (5 > 0) (5 > 10)
true
```

It takes practice to use these three reporters. Once you master them, you can create more complex IF statements rather than numerous simple ones.

You might use AND to check that a number is in the proper input range. For example

```
TO CHECK :NUMBER
IFELSE AND (:NUMBER > 0) (:NUMBER < 100)
   [PRINT [You are in the correct range.]]
   [PRINT [Pick a number between 0 and
   100.]]
END
```

OR can be useful if you want to write a game that controls the range of the turtle. For example

```
IF OR (XCOR > 139) (XCOR < -138) [RIGHT 180]
```

This statement will "bounce" the turtle off the sides of the page. (Remember, in Versions 1.0 and 1.1, you must use FIRST POS instead of XCOR.) Here's a more complex statement to keep the turtle on the page of an Apple screen.

```
IF (OR ((XCOR) > 138)
     ((XCOR) < -139)
     ((YCOR) > 89)
     ((YCOR) < -88))  [RIGHT 180]
```

If the turtle reaches the edge of the page, then it turns completely around.

There is yet another group of reporters. These are useful for checking input.

NUMBER?	returns true if its input is a number, otherwise false.
WORD?	returns true if its input is a word, otherwise false
LIST?	returns true if its input is a list, otherwise false.

For example, you might write a procedure like this:

```
TO DRAW.PICTURE
PRINT [Type a number:]
CHECK.AND.DRAW READWORD
END

TO CHECK.AND.DRAW :INPUT
IFELSE NUMBER? :INPUT
   [FORWARD :INPUT]
   [DRAW.PICTURE]
END
```

When you type DRAW.PICTURE, Logo prints

```
Type a number:
```

If you type a number, the turtle moves forward. If you type anything else, the procedure asks for a number again.

Another useful predicate that returns true or false is MEMBER? MEMBER? checks to see if its first input is contained in its second input. If so, it returns true. Otherwise it reports false. For example, if you want to check to see if a character is a vowel, you can use MEMBER?

```
TO CHECK :CHARACTER
IFELSE MEMBER? :CHARACTER [A E I O U]
   [PRINT [A vowel]]
   [PRINT [Not a vowel]]
END
```

If you type

```
CHECK "A
```

Logo displays

```
A vowel
```

but if you type

```
CHECK "B
```

Logo displays

```
Not a vowel
```

Suppose you wanted to turn some text on the page into Pig Latin. There are different versions of Pig Latin, but one set of rules says if a word begins with a vowel, then just add "ay"to the end; otherwise move the first letter to the end of the word and add "ay."

To turn the word into Pig Latin, you can use

```
TO PIG.WORD
SELECT
CF
IFELSE MEMBER? SELECTED [a e i o u]
    [UNSELECT
     SEARCH CHAR 32
     CB
     INSERT "ay]
    [CUT
     SEARCH CHAR 32
     CB
     INSERT CLIPBOARD
     INSERT "ay]
CF
END
```

Now all you need to do is move through the text until the end of the text is encountered..

TIPS AND TECHNIQUES

Earlier in this book you saw that bugs could arise in IF statements if you didn't use parentheses around conditions using =, > and <. You can avoid these problems by using EQUAL? instead of the = sign. Some versions of Logo also have GREATER? and LESS? primitives. LogoWriter does not, but you can write them.

```
TO GREATER :INPUT.1 :INPUT.2
OUTPUT :INPUT.1 > :INPUT.2
END

TO LESS :INPUT.1 :INPUT.2
OUTPUT :INPUT.1 < :INPUT.2
END
```

Using these predicates will avoid the problems created by the infix notation of standard mathematics.

The predicates AND, OR, and NOT are very powerful, but can be confusing to use at first. Recall that one technique for testing conditions is to run them in the Command Center. So, for example, if you have the statement

```
IFELSE AND (:NUMBER > 0) (:NUMBER < 100)
```

and it doesn't seem to be working correctly, you can test it by running it in the Command Center with specific numbers. For example, you might test

```
AND (5 > 0) (5 < 100)
AND (150 > 0) (150 < 100)
```

Studying the results of such tests will help you understand AND, OR, and NOT better, as well as help you debug your programs.

Writing your own predicates can be a way of simplifying your programs. Thus, if you have a statement such as

```
IF (OR ((XCOR) > 138)
      ((XCOR) < -139)
      ((YCOR) > 89)
      ((YCOR) < -88))  [RIGHT 180]
```

you can move the complex part of this statement to a procedure

```
TO EDGE?
OUTPUT (OR ((XCOR) > 138) ((XCOR) < -139)
   ((YCOR) > 89) ((YCOR) < -88))
END
```

and the IF statement them becomes

```
IF EDGE? [RIGHT 180]
```

making your program much easier to read and debug.

PROJECT SUGGESTIONS

Practice creating compound conditions using AND, OR, and NOT. Simply use SHOW statements to check the results of these conditions. The more you practice using these reporters, the easier it will be when you need them in a program. Can you write a procedure to check for a range of numbers of your choice? Can you keep the turtle within a "track" on the page? Where would NOT be useful?

Put DRAW.PICTURE into a drawing utility in which the user can input both distance and angle. Be sure to check that the user types a number for the angle.

Finish the Pig Latin program so that it will work for any text on the page.

38. FIRST, BUTFIRST, LAST, and BUTLAST

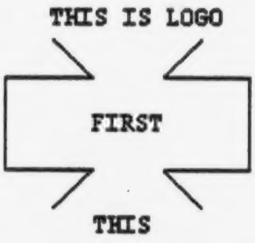
You have worked with text on the page, both by using the word processor and by writing procedures to manipulate text. Many of the commands you used to write procedures corresponded to key combinations. You have seen FIRST when you wrote a READWORD procedure. FIRST is one of the primitives in Logo that is used with what is traditionally called **list processing.** Unlike text processing in which you can manipulate text that you see on the page, list processing involves working with Logo objects found associated with local or global names. All of the primitives that you will examine in this and the following three chapters are reporters. Even though they are often referred to as list processing primitives, they can be used to manipulate any kind of Logo object: numbers, words, or lists.

The earliest versions of Logo did not contain a turtle at all. They had none of the elaborate features of modern versions of Logo like LogoWriter. List processing was at the center of any activities that were done with Logo. However, writing list processing programs is not always easy. It takes programming skill, understanding of reporters, and familiarity with recursion, including embedded recursion. These next several chapters merely introduce you to the primitives needed to do sophisticated list processing. If you are interested in pursuing these ideas, then you should move next to Brian Harvey's *Computer Science Logo Style* , *Volume 1* book mentioned in the Introduction.

Begin by studying these simple examples:

```
SHOW FIRST "LOGO
L

SHOW FIRST [THIS IS LOGO]
THIS

SHOW FIRST 54321
5
```

It should be clear that the reporter FIRST takes either a word or a list as input (a number is a special kind of word) and returns the first letter in the word or the first element in a list. You have diagrammed such reporters like this:



To be sure that you understand FIRST, what do you think Logo displays if you type

```
SHOW FIRST [[LIST ONE] [TWO] [AND THREE]]
```

Did you correctly deduce that Logo displays the first element of the list, which is the list [LIST ONE]?

There is also a reporter LAST. What do you think it does? What inputs does it need? What output does it produce? Can you state your thoughts clearly and precisely? Take a few moments to experiment with LAST.

There are two "companions" to FIRST and LAST. Study these examples.

```
SHOW BUTFIRST "LOGO
OGO

SHOW BUTFIRST [THIS IS LOGO]
IS LOGO

SHOW BUTFIRST 54321
4321
```

If BUTFIRST (short form BF) gets a word as input, it returns everything but the first letter of the word; similarly if it gets a list as input, it returns everything but the first element of the list.

As you might guess, there is also a BUTLAST reporter. BL is the short form. Can you describe precisely how it works? Can you predict what examples similar to the ones above will produce? Can you diagram the way it works? Experiment with BUTLAST.

These two pairs of reporters can be used to take lists apart by building a loop much as you used the idea of incrementing to change numerical values earlier in this book. Consider this procedure:

```
TO TAKE.APART :LIST.IN
PRINT FIRST :LIST.IN
TAKE.APART BUTFIRST :LIST.IN
END
```

If you type

```
TAKE.APART [THIS IS LOGO]
```

Logo prints

```
THIS
IS
LOGO
```

on the page and displays

```
FIRST DOESN'T LIKE [] AS INPUT IN
PRINT FIRST :LIST.IN
```

in the Command Center. FIRST will not accept an empty word or list as input. To avoid this problem and stop a procedure such as this, you can modify the last line as follows:

```
TO TAKE.APART :LIST.IN
PRINT FIRST :LIST.IN
IF NOT EMPTY? BUTFIRST :LIST.IN
   [TAKE.APART BUTFIRST :LIST.IN]
END
```

Recall that EMPTY? is a predicate that returns true if the word or list following it has nothing in it; false otherwise. Do you see why you must have the BUTFIRST in the last line rather than just :LIST.IN? Examine the diagram at the bottom of the page to see what is happening. It shows the changing values of the procedure input instead of the name of the procedure input.

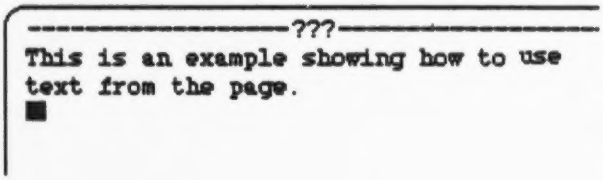
Take some time to be sure you understand what occurs in this procedure.

You will often seen a procedure like this written as follows.

```
TO TAKE.APART :LIST.IN
IF EMPTY? :LIST.IN [STOP]
PRINT FIRST :LIST.IN
TAKE.APART BUTFIRST :LIST.IN
END
```

This second version is a little easier to understand. It is not complicated by passing BUTFIRST :LIST.IN to the TAKE.APART. However, you need to be careful when you use this form that you don't overuse STOP. It is easy to find yourself with a procedure such as this with several STOP's in it and even some unwanted embedded recursion.

Although list processing primitives usually work with the values of names, like the TAKE.APART procedure above, you can use the text processing primitives you used earlier to transfer text on the page to a name within a procedure. Suppose you have

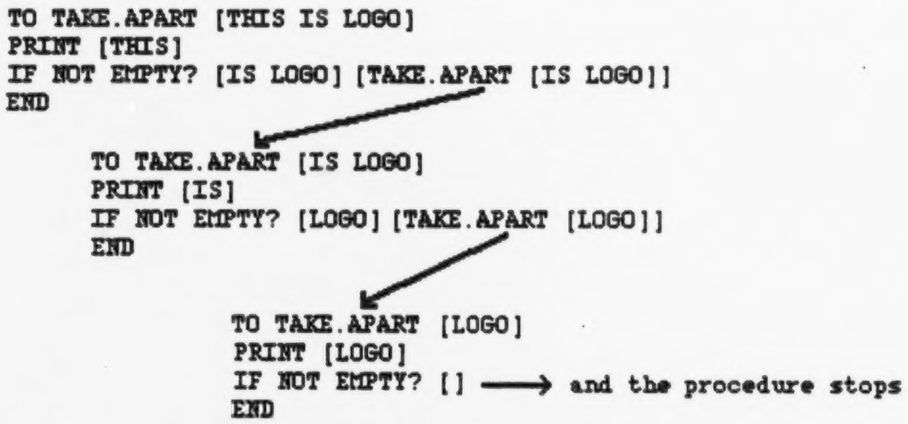


```
-------------------???-------------------
This is an example showing how to use
text from the page.
█
```

You can write

```
TO GET.TEXT
TOP
SELECT
BOTTOM
MAKE "TEXT PARSE SELECTED
PRINT []
TAKE.APART :TEXT
END
```

SELECTED reports everything up to the end of the page, in this case up to the period at the end of the sentence. SELECTED returns a word made up of all of the characters



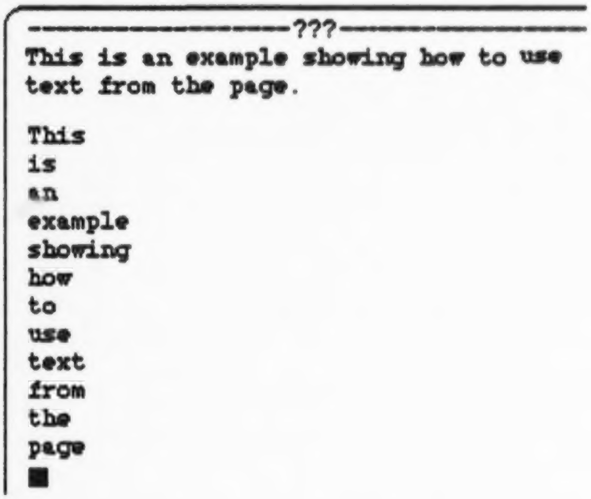
```
TO TAKE.APART [THIS IS LOGO]
PRINT [THIS]
IF NOT EMPTY? [IS LOGO] [TAKE.APART [IS LOGO]]
END

      TO TAKE.APART [IS LOGO]
      PRINT [IS]
      IF NOT EMPTY? [LOGO] [TAKE.APART [LOGO]]
      END

                TO TAKE.APART [LOGO]
                PRINT [LOGO]
                IF NOT EMPTY? [] ---> and the procedure stops
                END
```

selected; PARSE is a reporter that turns that "word" into a list. Then TAKE.APART puts one word of the text on each line.

Note that the MAKE is not necessary in this procedure. It could be written as

```
TO GET.TEXT
TOP
SELECT
BOTTOM
PRINT []
TAKE.APART PARSE SELECTED
END
```

When either of these procedures is run, you see



```
------------------???------------------
This is an example showing how to use
text from the page.

This
is
an
example
showing
how
to
use
text
from
the
page
■
```

TIPS AND TECHNIQUES

Thoroughly understanding the simple ideas presented in this chapter is the first step to moving into more advanced Logo programming. Beware, however, of the simplicity of the ideas. There can be subtle differences that are easy to miss. For example, did you notice that SELECTED returns a word? That means that if you type

```
SHOW SELECTED
```

you might see the *word*

```
This is an example
```

rather than the list

```
[This is an example]
```

All of the characters, including the spaces, make up one word. As you begin your work with list processing, it becomes very important to pay close attention to the distinction between Logo words and Logo lists.

PROJECT SUGGESTIONS

Spend some time experimenting with each of these new reporters as suggested above. Be sure that you understand how each functions. Pay particular attention to the difference between words and lists.

Diagram the TAKE.APART procedure using a word as input instead of a list.

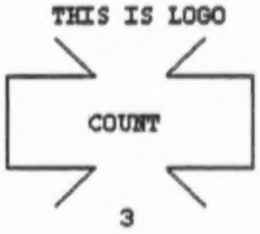
Modify the TAKE.APART procedure so that it lists the words in reverse order. Can you write a program that takes a sentence off the page using GET.TEXT and then displays it as a sentence in reverse order on the page?

39. ITEM and COUNT

You have seen the reporter TEXTLEN that returns the length of the text on the page. There is a similar list processing reporter that works on any word or list, not just the text on the page. Study these examples.

```
SHOW COUNT "LOGO
4
```

```
SHOW COUNT [THIS IS LOGO]
3
```



```
SHOW [A [B C] [D] [E F G]]
4
```

In the last example above, there are 4 elements in the list, the word A, and the lists [B C], [D] and [E F G]. Each counts as one Logo object in the list.

COUNT takes a word or list as input and returns the number of letters in the word or the number of elements in the list.

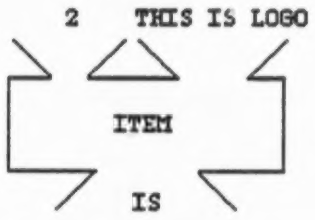
Yet another reporter allows you to examine a particular letter in a word or object in a list. Consider these examples.

```
SHOW ITEM 3 "LOGO
G
```

```
SHOW ITEM 2 [THIS IS LOGO]
IS
```

```
SHOW ITEM 3 [A [B C] [D] [E F G]]
[D]
```

ITEM takes two inputs: a number and either a word or a list. It returns the letter in the word or the element in the list in the place indicated by the number.



These new primitives can be put together to select a random element from a list. First, consider this statement:

```
PRINT RANDOM COUNT [THIS IS LOGO]
```

The possible numbers printed are 0, 1, and 2. Do you see why? (RANDOM reports numbers from 0 to 1 less than its input.) To pick randomly from a list, you can't use 0; ITEM needs a number between 1 and the number of objects in its input. This can be accomplished by using

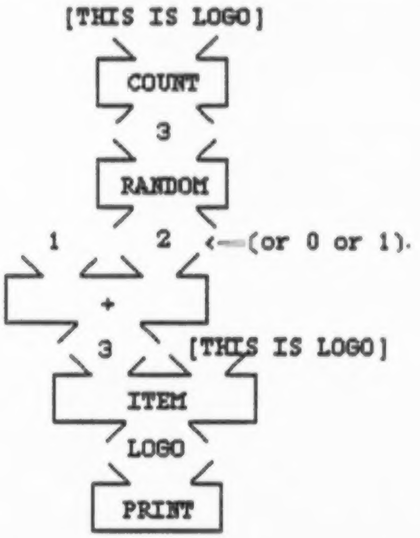
```
PRINT 1 + RANDOM COUNT [THIS IS LOGO]
```

This gives the numbers 1, 2, or 3.

Next, the problem is to pick the element represented by this number. ITEM should accomplish this.

```
PRINT ITEM (1 + RANDOM COUNT [THIS IS LOGO])
    [THIS IS LOGO]
```

Examine the statement above carefully. Study the diagram below. Look at one "box" at a time to see what is happening.



Learning to think through complex instructions like this is one of the keys in learning to use list processing reporters in Logo.

As it stands, the above example is not particularly useful. However, the idea can be put into a procedure that accepts an input.

```
TO PICK :IN
PRINT ITEM (1 + RANDOM COUNT :IN) :IN
END
```

If you type

```
PICK "LOGO
```

then one of the letters in LOGO is printed, and if you type

```
PICK [THIS IS LOGO]
```

then one of the three words in the list is printed, as shown in the above diagram.

However, to make PICK a tool that is useful in other procedures, the PRINT needs to be changed to OUTPUT.

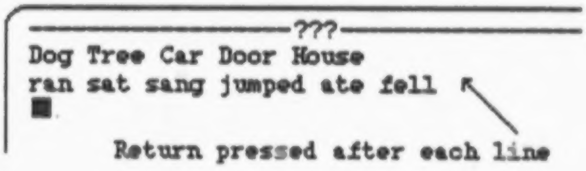
```
TO PICK :IN
OUTPUT ITEM (1 + RANDOM COUNT :IN) :IN
END
```

Now typing

```
PRINT PICK [THIS IS LOGO]
```

causes one of the three elements in the list to be printed on the page.

PICK can be used as a tool to generate a random sentence. Suppose that you type on the page



```
???
Dog Tree Car Door House
ran sat sang jumped ate fell
```

You can then use the text processing primitives that you learned earlier to get these words from the page.

```
TO GET.WORDS
TOP
SELECT
EOL
MAKE "NOUNS PARSE SELECTED
UNSELECT
CD
SOL
SELECT
EOL
MAKE "VERBS PARSE SELECTED
UNSELECT
PRINT []
(PRINT PICK :NOUNS PICK :VERBS)
END
```

Logo now writes a random sentence using the words from the page. Using the above words, Logo might produce

```
Dog ran
```

or

```
House jumped
```

or

```
Tree fell
```

and so forth.

Notice that MAKE was used in the above example. Here the two names NOUNS and VERBS were lists of "data" that might be used throughout a more complex program. This is an example where using MAKE seems appropriate.

TIPS AND TECHNIQUES

This PICK procedure is such a useful tool that many feel that it should be a primitive in Logo. In fact, the name PICK is used for this procedure in much that is written about Logo. With PICK you get the first real hint of Logo's extensibility. If a version of Logo doesn't have a particular primitive (XCOR, GREATER? PICK, etc.) you can just write it and add it to your page when needed. If you explore Logo programming in more depth, you will find yourself writing procedures that behave much like primitives. These procedures become your own personal "tool kit."

PROJECT SUGGESTIONS

Experiment with ITEM and COUNT. Can you find ways to use them in programs that you want to write? Be sure that you understand how these new reporters work.

Expand on the above sentence generator. Can you add articles, adjectives, and adverbs? Can you write a procedure to have the user add words to the page before the program started to make sentences? How about saving the words on a separate page?

Draw a diagram using inputs different from THIS IS LOGO for PRINT ITEM (1 + RANDOM COUNT :IN) :IN?

Write a program using PICK to deal a hand of cards. Use a very small, incomplete deck of cards as you create your program.

Can you write a MY.FIRST procedure that acts like FIRST, but doesn't use FIRST? How about a MY.BUTFIRST?

Here's a challenge: Write a MY.ITEM procedure that behaves just like ITEM, but doesn't use ITEM. Can you write a MY.COUNT?

40. WORD, SENTENCE, and LIST

So far you have used the list processing primitives to take lists and words apart, or to find one part of a list or word. There are times that you might also want to put words and/or lists together. This next group of reporters is used to accomplish this task.

Suppose that you want to add an S to a word to make it plural. You might try typing

```
(SHOW "DOG "S)
```

but Logo displays

```
DOG S
```

You can solve the problem with the reporter WORD. Typing

```
SHOW WORD "DOG "S
```

causes the word

```
DOGS
```

to be displayed. WORD takes two words as input and returns a single word.

The related reporter SENTENCE (short form SE) accepts two words or lists as inputs and returns a single list. Thus if you type

```
SHOW SENTENCE [THESE ARE] [TWO PARTS]
```

Logo displays

```
[THESE ARE TWO PARTS]
```

Recall that SHOW displays the brackets when you have it print a list. This property of SHOW can be very helpful when working with the list processing capabilities of Logo, especially when you are debugging programs. If you type

```
PRINT SENTENCE [THESE ARE]
    [TWO PARTS]
```

then you see

```
THESE ARE TWO PARTS
```

PRINT removes the outer pair of brackets.

In older Logo books, you will often see statements like this:

```
PRINT SENTENCE [HI,] :NAME
```

instead of

```
(PRINT [HI,] :NAME)
```

Many early versions of Logo did not allow you to use parentheses to allow PRINT to have more than one input, so you had to use SENTENCE to create a single input for PRINT.

LIST is another reporter that accepts two words or lists as input returns a list. For example, if you type

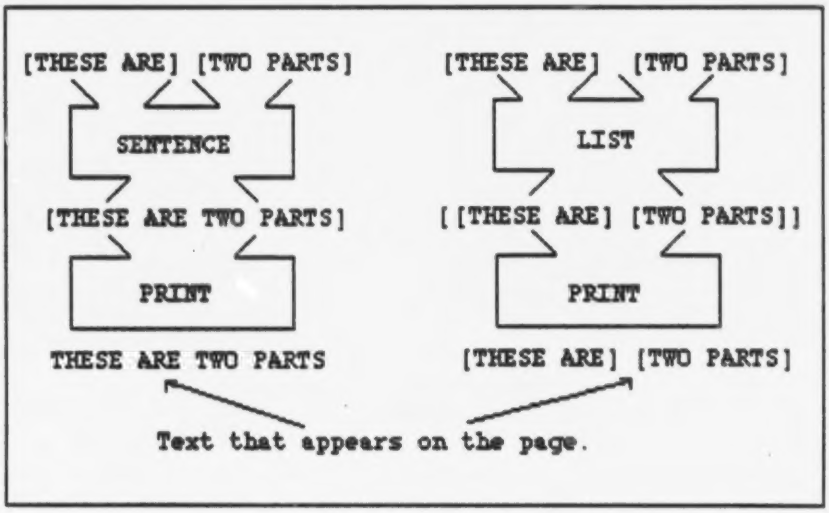
```
SHOW LIST [THESE ARE] [TWO PARTS]
```

Logo displays

```
[[THESE ARE] [TWO PARTS]]
```

Compare SENTENCE and LIST. Notice the difference in output with the same input. You may find the difference a bit confusing. The secret lies the fact that SENTENCE "flattens" a list; i.e., removes extra brackets.

Now, compare SENTENCE and LIST.



This technical distinction sometimes becomes important in more advanced work with lists.

One way you might use SENTENCE is to create a list from words typed at the keyboard.

```
TO GET.WORDS :THE.LIST
PRINT [Type a word..  Type S to stop.]
MAKE "THE.LIST SENTENCE :THE.LIST READ-
   WORD
IFELSE EQUAL? LAST :THE.LIST "S
   [USE.LIST :THE.LIST
   [GET.WORDS :THE.LIST]
END
```

The USE.LIST procedure, which is not shown here, would be a procedure that made some use of the list just generated.

The diagram at the bottom of the page follows the procedure GET.WORDS through several steps using the values for the name THE.LIST. As you looked at the diagram, did you see the bug in this program? Look carefully at the last call to USE.LIST. Do you see that the S is being passed to USE.LIST as one of the words in the list? Can you fix the bug?

TIPS AND TECHNIQUES

In this chapter you again see the importance of understanding the distinction between a Logo word and a Logo list. When you have trouble with SENTENCE and LIST, make use of SHOW to display the results. In more advanced programming, the distinction between using SENTENCE and LIST can be very important.

PROJECT SUGGESTIONS

Take the diagram of GET.WORDS and go one step further. That is, show the value of LAST :THE.LIST instead of just :THE.LIST.

Fix the bug in GET.WORDS. Then write a USE.LIST procedure. Perhaps you could have USE.LIST list the words on a single line on the page.

Take some time to explore the differences among WORD, SENTENCE, and LIST. Use SHOW with both simple and compound lists. See if you can predict the outcome of each statement before you press Return/Enter.

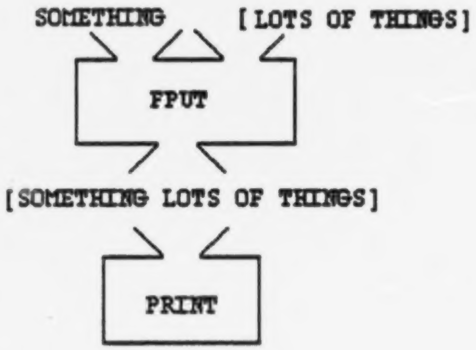
Use the ideas for creating random sentences and the ideas from this chapter for making lists of words and write a program that will accept lists of words from the keyboard and print random sentences on the page.

```
TO GET.WORDS [ ]
PRINT [Type a word..  Type S to stop.]
MAKE "THE.LIST SENTENCE [ ] READWORD <-- Cat is typed
IFELSE EQUAL? LAST [Cat] "S
 [USE.LIST [Cat] STOP] <--This line is not run.
 [GET.WORDS [Cat]]
END    TO GET.WORDS [Cat]
       PRINT [Type a word..  Type S to stop.]
       MAKE "THE.LIST SENTENCE [Cat] READWORD <-- Dog is typed
       IFELSE EQUAL? LAST [Cat Dog] "S
        [USE.LIST [Cat Dog] STOP] <-- This line is not run.
        [GET.WORDS [Cat Dog]
       END
             TO GET.WORDS [Cat Dog]
             PRINT [Type a word..  Type S to stop.]
             MAKE "THE.LIST SENTENCE [Cat Dog] READWORD <-- S is
             IFELSE EQUAL? LAST [Cat Dog S] "S                 typed
              [USE.LIST [Cat Dog S] STOP]
              [GET.WORDS [Cat Dog S]] <--Now this line is not run.
             END
```

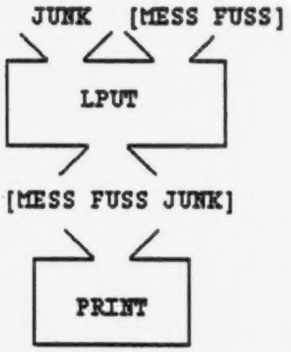
41. FPUT and LPUT

Sometimes you want to add things to an existing list rather than just putting lists together. The primitives to accomplish this task are LPUT and FPUT. Here are a couple of examples.

```
SHOW FPUT "SOMETHING [LOTS OF THINGS]
   [SOMETHING LOTS OF THINGS]
```



```
SHOW LPUT "JUNK [MESS FUSS]
[MESS FUSS JUNK]
```



FPUT accepts an object and a list as input and reports a list made up of the object and the list. LPUT accepts an object and a list as input and reports a list made up of the list with the first object on the end.

An example in the last chapter showed a way to create a list of words. Here is another approach.

```
TO ADD.NOUNS :LIST.IN
PRINT []
INSERT [(NEXT) NOUN ->]
ADD.MORE READWORD :LIST.IN
END

TO ADD.MORE :WORD.IN :THE.LIST
IF NOT (:WORD.IN = "QUIT)
   [ADD.NOUNS FPUT :WORD.IN :THE.LIST]
END
```

The second procedure is needed to "hold on" to the value typed so that it can be checked. The two procedures work together to produce a list that can be used. Below is an example of running this procedure.

```
ADD.NOUNS []

( NEXT ) NOUN - - >CAT
( NEXT ) NOUN - - >DOG
( NEXT ) NOUN - - >RAT
( NEXT ) NOUN - - >QUIT
```

Notice that the list THE.LIST in procedure ADD.MORE is a local name. That is, it only exists when ADD.MORE is running. If you need to use the list elsewhere, then you must pass it to another procedure. For example, you might modify ADD.MORE as follows:

```
TO ADD.MORE :WORD.IN :THE.LIST
IFELSE (:WORD.IN = "QUIT)
   [USE.LIST :THE.LIST]
   [ADD.NOUNS FPUT :WORD.IN :THE.LIST]
END
```

Procedure USE.LIST then makes use of the list created.

FPUT or LPUT can also be used to put together several paragraphs from the page into one list. For example,

```
TO GET.TEXT
TOP
SELECT
EOL
GET.MORE (LIST PARSE SELECTED)
END
```

```
TO GET.MORE :ALL
CF
UNSELECT
SELECT
EOL
IFELSE TEXTLEN = TEXTPOS
   [PRINT :ALL]
   [GET.MORE LPUT PARSE SELECTED :ALL]
END
```

The procedure GET.TEXT selects the first paragraph and passes a list made up of the text in the first paragraph into ALL in GET.MORE. Then each subsequent paragraph is selected and added to the end of the list.

As with the other list processing primitives, you should spend some time working with FPUT and LPUT. Be sure you understand how they work. SHOW is a good vehicle for being sure whether you are working with words or lists.

TIPS AND TECHNIQUES

You have now seen the list processing primitives that are necessary for advanced programming in Logo. Have you noticed that they fall into several categories?

FIRST, LAST, BUTFIRST, BUTLAST: used to take Logo objects apart
ITEM, COUNT: used to examine Logo objects
WORD, SENTENCE, LIST: used to put together Logo objects
FPUT, LPUT: used to add Logo objects to a list

Keeping these categories in mind is helpful as you think about writing a program that manipulates Logo objects.

PROJECT SUGGESTIONS

Draw a diagram showing what happens when ADD.NOUNS is run with the words shown in the sample run in the text.

Draw a diagram showing what is happening when you run GET.TEXT.

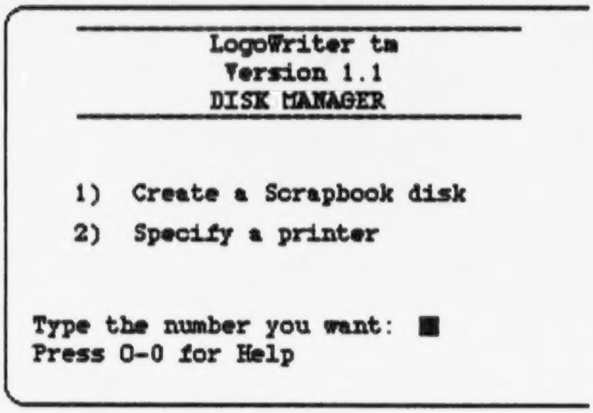
The primitive FLIP causes the page to be flipped. The reporter FRONT? tells you whether you are on the front of the page or not. Can you write a program that will write a procedure on the flip side of the page? Can you allow the user to enter lines of code that then get put into a procedure?

Appendix 1: Setting up your LogoWriter Disks

Getting Started Using LogoWriter 1.0 or 1.1

If you are using **LogoWriter version 1.0 or 1.1**, then the disk which you received has two sides. One side contains the LogoWriter language; the other is a Disk Manager (which may be labelled Master Disk) which will allow you to create Scrapbook (file) disks and configure your LogoWriter disk for the appropriate printer.

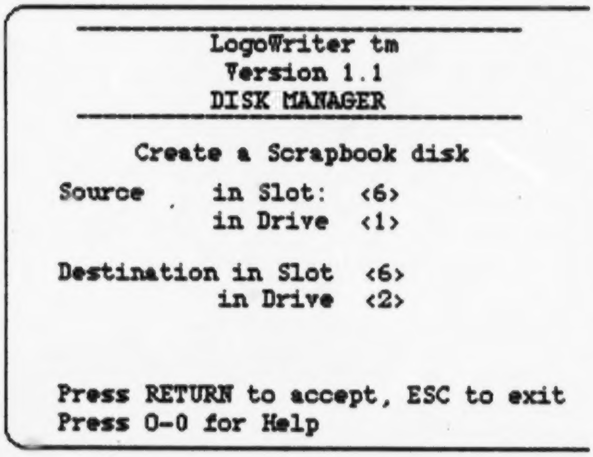
First, start the LogoWriter Disk Manager. After the initial LogoWriter screen, you should see



```
          LogoWriter tm
           Version 1.1
          DISK MANAGER

  1)  Create a Scrapbook disk
  2)  Specify a printer

Type the number you want: █
Press O-O for Help
```

Select 1.

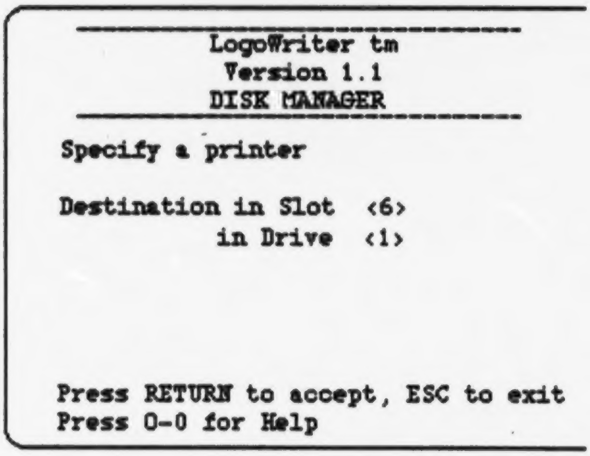
You will then see



```
          LogoWriter tm
           Version 1.1
          DISK MANAGER
     Create a Scrapbook disk
Source      in Slot:  <6>
            in Drive  <1>

Destination in Slot   <6>
            in Drive  <2>

Press RETURN to accept, ESC to exit
Press O-O for Help
```

You must press Return to accept the slot and drive number; the arrow keys don't work. If you make a mistake, press Esc and try again. If your disk drive is not connected to slot 6, change the slot number. If you have only one disk drive, change the drive number from <2> to <1>. Press Return and follow the directions at the bottom of the screen. When the message WORK COMPLETED appears, remove your new Scrapbook disk. Repeat the process until you have the number of Scrapbook disks that you feel you will need.

Next, you need to configure your LogoWriter disk for the appropriate printer. Begin by returning to the main menu (the first screen shown above) and pressing 2. A list of printers will appear. Use the arrow keys to select the appropriate printer and Return to accept that printer. Next, a list of interface cards appears. Again use the arrow keys to select the appropriate card and press Return. Now this menu appears:



```
          LogoWriter tm
           Version 1.1
          DISK MANAGER
Specify a printer

Destination in Slot  <6>
            in Drive <1>




Press RETURN to accept, ESC to exit
Press O-O for Help
```

Select the correct slot and drive, put in your LogoWriter disk (not a Scrapbook disk and not the Disk Manager) and press Return. When WORK COMPLETED appears, you are finished.

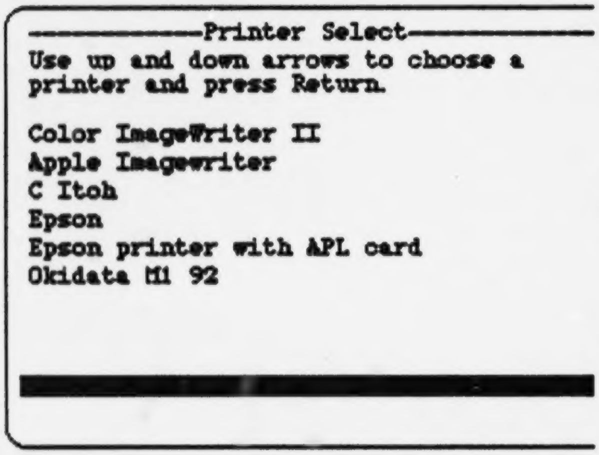
Getting Started Using LogoWriter 2.0

If your are using a recent version of LogoWriter, then you do not need to use a master disk to configure your LogoWriter master disk for a printer. You can find out the version number by typing

SHOW .VERSION

Don't forget the period before the word "version." If you have an Apple version with number lower than 2.03, call LCSI (1-800-321-5646) for a free upgrade.

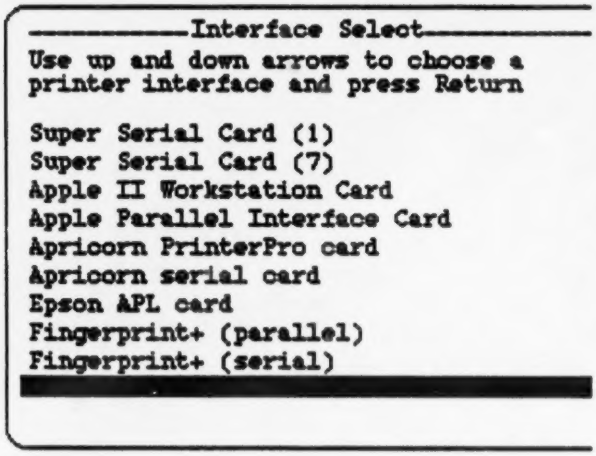
To get started with the Apple IIe/IIc version, put your LogoWriter language disk in the disk drive and start your computer. Immediately hold down the "Open Apple" key. After a good bit of whirring by the disk drive, a screen allowing you to choose a printer appears. If you are using an Apple computer, you see:



```
------------Printer Select-------------
Use up and down arrows to choose a
printer and press Return.

Color ImageWriter II
Apple Imagewriter
C Itoh
Epson
Epson printer with APL card
Okidata Ml 92
```

Use the arrow keys to select the printer you are using. Then press Return/Enter.

Next, a menu that allows you to select an interface card appears. Again, if you are using an Apple IIe or IIc computer, you see:



```
-----------Interface Select-----------
Use up and down arrows to choose a
printer interface and press Return

Super Serial Card (1)
Super Serial Card (7)
Apple II Workstation Card
Apple Parallel Interface Card
Apricorn PrinterPro card
Apricorn serial card
Epson APL card
Fingerprint+ (parallel)
Fingerprint+ (serial)
```

In some cases this list is quite long. You can scroll down the list using the arrow keys. When you find the interface card for your computer, press Return/Enter. After some more whirring by the disk drive, you will see the usual LogoWriter introductory screen. The disk in your disk drive is now configured for the printer and card that you selected.

If you are using the Apple GS version of LogoWriter, start LogoWriter and select the page SETUP.PRINTER. Type the word associated with your printer to configure your disk. If you are using the built-in printer port or your printer card is in slot 1, press Esc to return to the Contents Page. If you have a printer card in a different slot, type SETPRINTERSLOT *slotnumber* to change the slot number used.

If you are using IBM LogoWriter version 2.0, then there are a number of batch files that you run from DOS to set up your printer. You'll need to see your documentation to know which file to use for your particular combination of interface card and printer.

Checking Your Printer

Finally, you need to check your newly configured disk to see if it prints correctly. When you have the LogoWrite introductory page on the screen, press Return, and select NEW PAGE from the Contents page.

Now type

REPEAT 4 [FORWARD 50 RIGHT 90]

A square should appear on your screen. Next press the UP keys. Then type several lines of text without pressing Return. For example, you might type

```
This is a test to see if my printer is
working correctly.  If your printer
configuration is not working correctly,
you can call for technical assistance.
```

Now, press the DOWN keys. Type PRINTSCREEN. A picture should appear including the text you typed (in large letters) and the square. Next type PRINTTEXT. The text should appear exactly as it is on the screen (in 40 columns), but double spaced. Finally; type PRINTTEXT80. The text should now appear double spaced, but 60 characters wide.

If any of the above tests does not work correctly, check your configuration again. If nothing is wrong, call LogoWriter technical support at 514-331-2791.

APPENDIX 2: STARTUP

STARtUP is a word that has special meaning in LogoWriter. There are two uses for the word STARTUP: to name procedures and to name pages.

If you name a procedure STARTUP, then that procedure will automatically run when you load the page with procedure STARTUP on the flip side.

If you name a page STARTUP, then that page is automatically loaded when you start LogoWriter.

Thus, if you want a page to automatically load and run, then you name the page STARTUP. On the flip side of this page, you must have your toplevel procedure called from a procedure named STARTUP.

NOTE: There can be only one STARTUP page on a scrapbook disk and only one STARTUP procedure on the flip side of any page.

APPENDIX 3: Chaining Pages

If you have written programs that are very long, you may have discovered that you can use up the space that is available in LogoWriter. However, the space available on one page is really not a limitation. You can put LogoWriter pages together to get more space, just as you can add pages to a scrapbook with paper pages.

Begin by creating two pages:

DESIGN should contain some graphics design

TEXT should contain some writing on the page.

Now create a third page called BOTH. Flip the page and enter the following procedure:

```
TO CHOOSE
GETPAGE "DESIGN
WAIT 20
GETPAGE "TEXT
END
```

Type

```
CHOOSE
```

and you first see your design and then your text. When the procedure is done, you are on the page TEXT. You must include a command to get the page BOTH in order to end up on BOTH when the procedure finishes running. You can make this procedure interactive by adding a few lines:

```
TO CHOOSE
GETPAGE "DESIGN
WAIT 20
GETPAGE "TEXT
WAIT 20
GP "BOTH
INSERT [Would you like to see these pages
   again?]
IF READLIST = [Y] [CHOOSE]
END
```

Notice that READLIST is checked against the list containing only the letter Y. If the user types YES, the procedure stops. It is usually a good idea to tell the user what to type. For example,

```
INSERT [Would you like to see these pages
   again? (Y or N)]
```

LogoWriter has a special word, STARTUP, (See Appendix 2) which can make working with multiple pages even more interesting. Change your DESIGN page so that it contains these procedures

```
TO STARTUP
MAKE.DESIGN
ASK.Q
END

TO MAKE.DESIGN
CG
 <any design you want here>
END
```

```
TO ASK.Q
SHOW [Do you want to see the design again?]
IF READLISTCC = [Y] [STARTUP]
END
```

Notice that ASK.Q calls STARTUP recursively. This recursion is delayed, but it is recursion nonetheless. Save this page.

Now modify your page TEXT so that it contains

```
TO STARTUP
MESSAGE
ASK.Q
END

TO MESSAGE
CT
<any text you want goes here>
END

TO ASK.Q
SHOW [Do you want to see the message again?]
IF READLISTCC = [Y] [STARTUP]
END
```

Save this page. Now, go to your page BOTH, and type

```
CHOOSE
```

Logo gets the page DESIGN and the STARTUP procedure is automatically run. Anytime you put a STARTUP procedure on a page, that procedure runs automatically every time you load the page. So, when DESIGN is loaded from BOTH, the design is repeated as often as you want. Then Logo gets the page TEXT and its STARTUP procedure is automatically run. The message is repeated as often as you want. Then Logo returns to the page BOTH and asks if you want to repeat the entire program again.

When you set up both a STARTUP page and STARTUP procedures, you may find it hard to stop the procedures from running when you want to change them. To get any STARTUP procedure to stop, hold down the stop keys as soon as the page starts to load. Don't just press them and then release them. Hold them down until the message "stopped!" appears at the bottom of the screen.

APPENDIX 4: Using Tools

There are a number of commands that allow you to work with "tool" pages in LogoWriter. A tool page is really no different from any other LogoWriter page. However, when you use a page as a tool page, the procedures from that page are loaded into the memory of the computer. They are not visible on the flip side of the page. Thus they use up less space. To use a page as a tool page, you type

```
GETTOOLS "page.name
```

The procedures on the page loaded then become available on the current page. To see what tools are available, you can type

```
PRINT TOOLLIST
```

TOOLLIST is a reporter that returns the list of available tool procedures. Although you can't see the procedures, you can at least see the names. If you no longer want tools, you can type

```
CLEARTOOLS
```

When you are using tool pages, it is easy to forget how many procedures you have added to your page using GETTOOLS. If you are working with tools and get an "out of space" error, try using CLEARTOOLS to reclaim some memory space.

APPENDIX 5: The WHEN Command

The WHEN command in LogoWriter allows you to "program" keys to perform a specific task. This command is useful with the word processing mode of LogoWriter or when writing games, but any LogoWriter commands can be used with WHEN.

What are the things you commonly put in a letter? Usually there is a date, a greeting, and a closing. On the back of the page, type

```
TO DATE
WHEN "X [PRINT [January 5, 1987]]
END
```

Be careful to put the single quotation mark (") before the X with no space. Now flip the page and type

```
DATE
```

Nothing happens. Next, press Control/X. Be sure to hold down the Control Key and then press the X key. What happens? Press Control/X again. Do you see that Control/X now stands for the date? Add the following procedures to the flip side of the page:

```
TO LETTER.HELPERS
DATE
GREETING
CLOSING
END
```

```
TO GREETING
WHEN "Y [PRINT [] PRINT [Dear]]
END

TO CLOSING
WHEN "Z [PRINT [Sincerely,]
         REPEAT 4 [PRINT []]
         PRINT [T. H. E. Turtle]]

END
```

Now type

```
LETTER.HELPERS
```

and try typing Control/X, then Control/Y, writing a short letter, and finally Control/Z. Now all you have to do is type

```
PRINTTEXT80
```

and your letter is done!

There are 10 keys that can be programmed using WHEN. They are

N O P Q R V W X Y and Z

You can have up to 10 different events active at the same time.

When you run a procedure containing WHEN, you create an "event." In order to get rid of the programming of these special keys you can type

```
CLEAREVENTS
```

This command removes all programming from the keys until a procedure containing WHEN is run again.

APPENDIX 6: ProDOS AND IBM DOS Primitives

LogoWriter Version 2.0 uses the ProDOS operating system on the Apple or the IBM DOS operation system on IBM and IBM compatible computers. There are a number of LogoWriter primitives that allow you to use DOS from within LogoWriter. There are discussed briefly below.

To understand many of these primitives, you must understand **pathnames**. A path name consists of a disk name, followed by any number of subdirectory names, followed by a file name. Disks can be given names. The pages that you put on a disk using LogoWriter are DOS **files**. If you want to group several files together, you can accomplish this by putting them in a **directory**. Using directories in LogoWriter is particularly useful. You can have a different SHAPES page in each directory allowing you to have as many SHAPES pages on a disk as you wish!

In ProDOS, the "/" character is used to separate the parts of a pathname; in IBM DOS the "\" character is used; and in LogoWriter, the "%" symbol is used. So, in a ProDOS environment a path name might be

/MY.DISK/A.DIRECTORY/MY.FILE

while in IBM DOS the path name might be

\MY.DISK\A.DIRECTORY\MY.FILE

In LogoWriter you would use

%MY.DISK%A.DIRECTORY%MY.FILE

(Note: The GS Version uses the "/" character in path names.)

BYE (ProDOS only)

This primitive lets you leave LogoWriter completely to go to another application. It automatically causes the page to be saved.

CHDIR *name/pathname* (IBM only)

Changes the directory for saving and loading pages.

COPYFILE *name/pathname1 name/pathname2*

This primitive allows you to make a copy of any page (file) from the Command Center. Thus you might type

```
COPYFILE "SQUARES "MANY.SQUARES
```

After this statement is run, you have two copies of the page SQUARE, one named SQUARE and the other named MANY.SQUARES.

CREATEDIR *pathname* (Apple only)

This command is used to create a directory (or subdirectory) on your disk. You might type

```
CREATDIR "%MY.DISK%MY.PAGES
```

Now there is a directory on the disk entitled MY.PAGES. This directory can have its own SHAPES page and you can save pages within that directory. To copy a file into this directory, you might type

```
COPYFILE "%MY.DISK%SQUARES
    "%MY.DISK%MY.PAGES%MY.SQUARES
```

Now there is a copy of the page SQUARES that is found on the list in the Contents page when you start LogoWriter in the MY.PAGES directory with the name MY.SQUARES.

CURRENTDIR (IBM ONLY)

Reports the name of the current directory.

DIRECTORIES

DIRECTORIES is a reporter that returns a list of the current subdirectories on the disk. You can type

```
SHOW DIRECTORIES
```

to see a list of the directories in the Command Center.

DOS (IBM only)

Saves the page and leaves LogoWriter.

ERASEFILE *name/pathname*

This command allows you to remove a page from any directory on the disk. It is equivalent to using Erase to End of Line on the Contents page.

FILELIST

FILELIST is a reporter that returns a list of all of the files on the current disk. This includes **all** files, not just LogoWriter pages. (See Appendix 7)

FILELIST *name* (GS Version)

FILETYPE reports the type of the named file, (e.g. "page","text", "shapes", etc.

LOADPIC *name/pathname*

LOADPIC allows you to load any standardly saved ProDOS graphics image to be loaded onto a LogoWriter page. (See Appendix 7)

LOADTEXT *name/pathname*

LOADTEXT allows you to load any standardly saved ProDOS ASCII text file to be loaded onto a LogoWriter page. (See Appendix 7)

MKDIR *name/pathname* (IBM only)

Creates a subdirectory. This works like CREATEDIR. See above.

ONLINE (Apple only)

This reporter returns the names of the volumes (disks) that are currently available. If you type

```
SHOW ONLINE
```

a list of the names of the disks in the disk drive is reported. Note that they are in reverse order, i.e., Drive 1 is shown last. (The GS Version also reports the number of devices connected to the computer.)

PREFIX (Apple only)

This reporter returns the current volume or directory name. That is, it gives you the path name except for the page/file name.

RENAME *name/pathname1 name/pathname2*

This command allows you to rename any file on disk. You might type

```
RENAME "%MY.DISK%MY.DIR%MY.FILE
   "%MY.DISK%MY.DIR%YOUR.FILE
```

to give change the name of the file MY.FILE to YOUR.FILE. Note that MY.FILE is on disk MY.DISK and in subdirectory MY.DIR.

RMDIR *name/pathname* (IBM only)

Removes a directory from disk

SAVEPIC *name/pathname*

The command allows you to save the image on the screen. The image is saved as a standard ProDOS graphics image. This is the image you see when you use PRINTSCREEN. (See Appendix 7)

SAVETEXT

This command saves any text on the page currently showing as an ASCII file. Note that it saves the text on the front of the page if that is showing when SAVETEXT is typed, and saves the procedures only if the flip side is showing when SAVETEXT is typed.

SETPREFIX *pathname* (Apple only)

This command allows you to change the current prefix (see PREFIX).

SETSLOT *number* (Apple IIe/IIc only)

This command allows you to set the slot for the disk drive interface card.

SLOT (Apple IIe/IIc only)

This reporter returns the current slot for the disk drive interface card.

.VERSION

Reports the version number in most recent versions of Logowriter.

For more information on using subdirectories in LogoWriter, see Eadie Adamson's column *Logo Ideas* in the September, 1988 issue of the *Logo Exchange*.

APPENDIX 7: Exchanging Files

LogoWriter Version 2.0 saves standard DOS graphics and text files. This means that LogoWriter files can be used with other DOS applications and that files from other applications can be used with LogoWriter.

Any application, such as a word processor, that can save standard ASCII (text) files can create files that can be used with LogoWriter. To move text from your word processor to LogoWriter,

1. Save your document from your word processor as a text or ASCII file.

2. Use the LOADTEXT command (See Appendix 6) to load the text into LogoWriter.

3. Save your LogoWriter page.

To move text from LogoWriter to another application such as a word processor:

1. Save the text on your page using SAVETEXT (See Appendix 6)

2. Start your other application and load the text file.

3. Save the document in your other application.

Note that for each of these steps, you will have to use the fill pathname in LogoWriter (See Appendix 6)

Graphics images can also be moved between LogoWriter and other applications. To move a graphics image from another application to LogoWriter,

1. Save your document from your other application. (You will need to check your documentation to be sure that it is saved as a standard file.)

2. Use the LOADPIC command (See Appendix 6) to load the text into LogoWriter.

3. Save your LogoWriter page.

To move a graphics image from LogoWriter to another application,

1. Save the text on your page using SAVEPIC (See Appendix 6)

2. Start your other application and load the graphics file.

3. Save the document in your other application.

Again, you will have to use the file pathname in LogoWriter. For more information on transporting text and graphics between LogoWriter and other applications, see *Logo Ideas* in the October 1988 and December 1988 issues of the *Logo Exchange*, published by International Society for Technology in Education (ISTE).

APPENDIX 8: LogoWriter Keys

	APPLE	IBM	COMMODORE
General Purpose Keys			
Flip Flips a page over	Apple-F	Ctrl-F	Commodore-F
Up Moves the cursor up from Command Center	Apple-U	Ctrl-U	Commodore-U
Down Moves cursor down into the Command Center	Apple-D	Ctrl-D	Commodore-D
Esc Used to leave a page or a special mode	Esc	Esc	Esc
Stop Stops program or instruction; returns to Command Center (See also Esc.)	Apple-S	Ctrl-Break	Commodore-S
Help Gets the Help page	Apple-0	Fn-10	Commodore-0
Contents Page Keys			
Up and Down Arrows Moves cursor up and down through page names that are in the scrapbook	Up & Down Arrows	Up & Down Arrows	Up & Down Arrows
Top of Page Moves the cursor to the top of the Contents list	Apple-up arrow	Home	Commodore up arrow
Bottom of Page Moves the cursor to the last page name in the Contents list	Apple-down arrow	End	Commodore down arrow
Return Enter Chooses the page that the cursor is on.	Return	Enter	Return
Erase to end of Line Erases the page named on the line that the cursor is on – permanently!	Apple-6	Fn-6	Commodore-6

Graphics Keys

Turtle-Move Makes it possible to move the turtle with the arrow keys; Esc to leave	Apple-9	Fn-9	Commodore-9
Label Text typed is added to the picture; use arrow keys to move; Esc to leave	Apple-8	Fn-8	Commodore-8

Word Processing Keys

Select Starts select mode; use arrows to select	Apple-1	Fn-1	Commodore-1
Cut Removes selected text; puts on the clipboard	Apple-2	Fn-2	Commodore-2
Copy Puts copy of selected text on the clipboard	Apple-3	Fn-3	Commodore-3
Paste Puts contents of the clipboard at the cursor position	Apple-4	Fn-4	Commodore-4
Erase to End of Line Erases all text cursor from cursor to the end of line	Apple-6	Fn-6	Commodore-6
Next Screen Displays the next screen full of text	Apple ->	Pg-Dn	Commodore->
Previous Screen Displays the previous screen of text	Apple <-	Pg-Up	Commodore<-
Top of Page Moves the cursor to the beginning of the text on the page	Apple-Up arrow	Home	Commodore-up arrow
Bottom of Page Moves cursor to the end of the text on the page	Apple-down arrow	End	Commodore-down arrow
Beginning of Line Moves the cursor to beginning of the line the cursor is on	Apple-B	Ctrl <-	Commodore-B
End of Line Moves the cursor to the end of the line; to where the next Return Enter was typed	Apple-E	Ctrl ->	Commodore-E
Open a line Opens new line after current cursor position	Apple-O	Insert	Commodore-O
Delete to the right Deletes the character under the cursor	Control-D		

Shapes Page Keys

Esc Returns you to where you were when Shapes was chosen	Esc	Esc	Esc
Flip Switches between the front (all shapes) and individual shapes	Apple-F	Ctrl-F	Commodore-F

On the flip side of the page only:

Next Screen Displays the next shape for editing	Apple ->	Pg-Dn	Commodore->
Previous Screen Displays the previous shape for editing	Apple <-	Pg-Up	Commodore<-
Arrow keys Moves the shape cursor around the shape	Arrow keys	Arrow keys	Arrow keys
Space Bar Empties or fills a space within a shape	Space bar	Space bar	Space bar
Cut Clears a shape and stores it in memory	Apple-2	Fn-2	Commodore-2
Copy Stores a copy in memory without erasing it.	Apple-3	Fn-3	Commodore-3
Paste Puts a shape that been placed in memory into the shape that is displayed	Apple-4	Fn-4	Commodore-4
Stop Cancels editing; restores to shape there before editing began	Apple-S	Ctrl-Break	Commodore-S

SUMMARY OF LOGOWRITER KEYS FOR THE APPLE IIe/IIc

General Purpose Keys

Flip	Apple-F
Up	Apple-U
Down	Apple-D
Quit, leave page	Esc
Stop	Apple-S
Help	Apple-0

Word Processing Keys

Select	Apple-1
Cut	Apple-2
Copy	Apple-3
Paste	Apple-4
Erase to End of Line	Apple-6
Next Screen	Apple ->
Previous Screen	Apple <-
Top of Page	Apple-Up Arrow
Bottom of Page	Apple-Down Arrow
Beginning of Line	Apple-B
End of Line	Apple-E
Open a line	Apple-O
Delete right	Control-D

Shapes Page Keys

Quit, leave page	Esc
Flip	Apple-F

On the flip side of the page only:

Next Screen (Next Shape)	Apple ->
Previous Screen (Previous Shape)	Apple <-
Up, Down, Left, Right	Arrow keys
Cut	Apple-2
Copy	Apple-3
Paste	Apple-4
Make or Erase Block	Space Bar

Graphics Keys

Turtle-Move	Apple-9
Label	Apple-8
Leave Label or Turtle-Move	Esc

Contents Keys

Top of Page	Apple-Up Arrow
Bottom of Page	Apple-Down Arrow
Remove a Page (Erase to end of line)	Apple-6

SUMMARY OF LOGOWRITER KEYS FOR THE IBM

General Purpose Keys

Flip	Control-F
Up	Control-U
Down	Control-D
Quit, leave page	Esc
Stop	Control-Break
Help	Fn-0

Word Processing Keys

Select	Fn-1
Cut	Fn-2
Copy	Fn-3
Paste	Fn-4
Erase to End of Line	Fn-6
Next Screen	PgDn
Previous Screen	PgUp
Top of Page	Home
Bottom of Page	End
Beginning of Line	Control <-
End of Line	Control ->
Open a line	Insert

Shapes Page Keys

Quit, leave page	Esc
Flip	Control-F

On the flip side of the page only:

Next Screen (Next Shape)	PgDn
Previous Screen (Previous Shape)	PgUp
Up, Down, Left, Right	Arrow keys
Cut	Fn-2
Copy	Fn-3
Paste	Fn-4
Make or Erase Block	Space Bar

Graphics Keys

Turtle-Move	Fn-9
Label	Fn-8
Leave Label or Turtle-Move	Esc

Contents Keys

Top of Page	Home
Bottom of Page	End
Remove a Page (Erase to end of line)	Fn-6

SUMMARY OF LOGOWRITER KEYS FOR THE COMMODORE

General Purpose Keys

Flip	Commodore-F
Up	Commodore-U
Down	Commodore-D
Quit, leave page	Esc
Stop	Commodore-S
Help	Commodore-0

Word Processing Keys

Select	Commodore-1
Cut	Commodore-2
Copy	Commodore-3
Paste	Commodore-4
Erase to End of Line	Commodore-6
Next Screen	Commodore ->
Previous Screen	Commodore <-
Top of Page	Commodore-Up Arrow
Bottom of Page	Commodore-Down Arrow
Beginning of Line	Commodore-B
End of Line	Commodore-E

Shapes Page Keys

Quit, leave page	Esc
Flip	Commodore-F

On the flip side of the page only:

Next Screen (Next Shape)	Commodore ->
Previous Screen (Previous Shape)	Commodore <-
Up, Down, Left, Right	Arrow keys
Cut	Commodore-2
Copy	Commodore-3
Paste	Commodore-4
Make or Erase Block	Space Bar

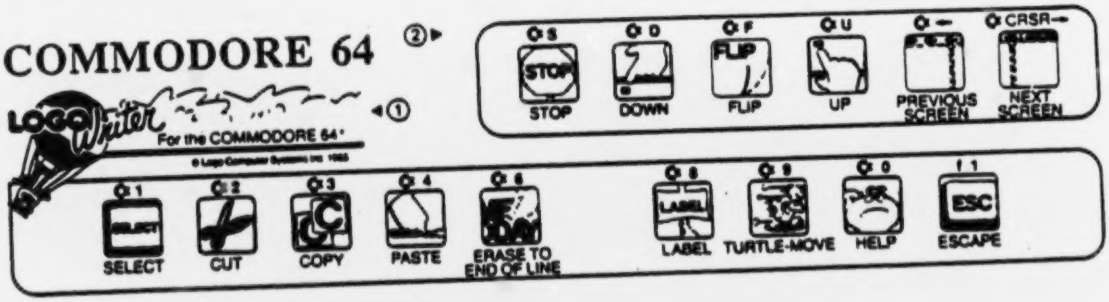
Graphics Keys

Turtle-Move	Commodore-9
Label	Commodore-8
Leave Label or Turtle-Move	Esc

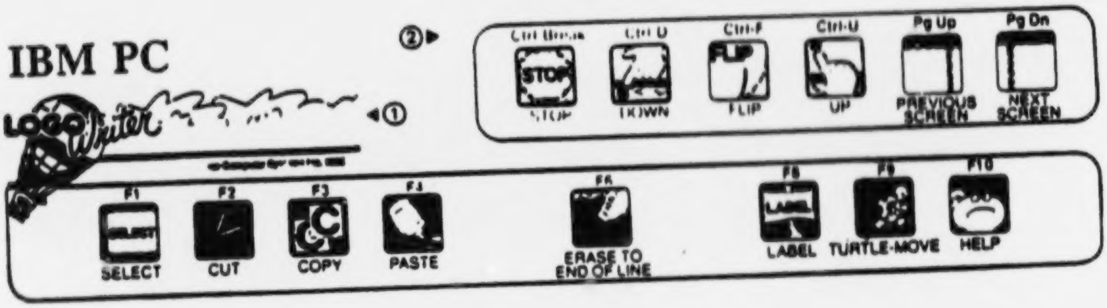
Contents Keys

Top of Page	Commodore-Up Arrow
Bottom of Page	Commodore-Down Arrow
Remove a Page (Erase to end of line)	Commodore-6

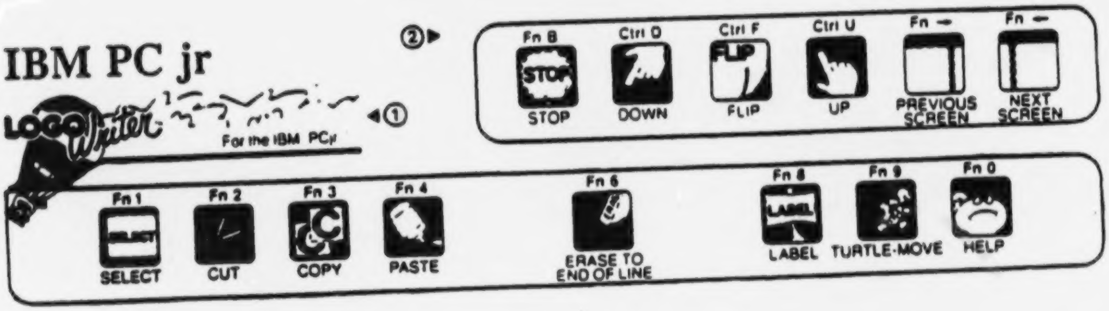
COMMODORE 64


②►
S
STOP
STOP
D
DOWN
F
FLIP
FLIP
U
UP
←
PREVIOUS SCREEN
CRSR→
NEXT SCREEN
LOGO Writer
For the COMMODORE 64
◄①
1
SELECT
SELECT
2
CUT
3
C
COPY
4
PASTE
6
ERASE TO END OF LINE
8
LABEL
LABEL
9
TURTLE-MOVE
0
HELP
f1
ESC
ESCAPE

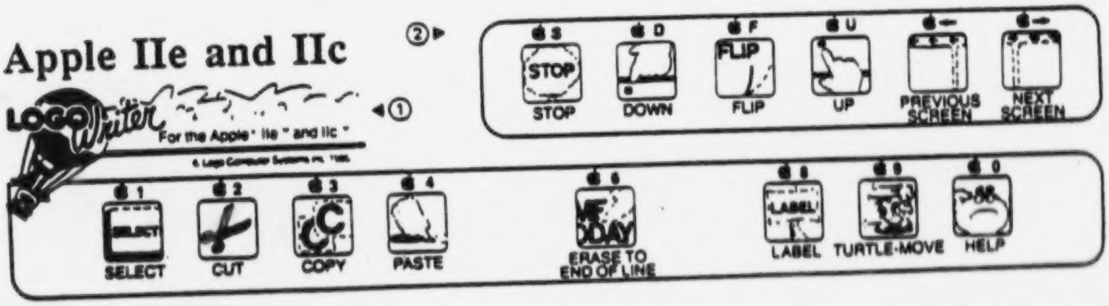
IBM PC


②►
Ctrl Break
STOP
STOP
Ctrl D
DOWN
Ctrl-F
FLIP
FLIP
Ctrl-U
UP
Pg Up
PREVIOUS SCREEN
Pg Dn
NEXT SCREEN
LOGO Writer
◄①
F1
SELECT
SELECT
F2
CUT
F3
C
COPY
F4
PASTE
F6
ERASE TO END OF LINE
F8
LABEL
LABEL
F9
TURTLE-MOVE
F10
HELP

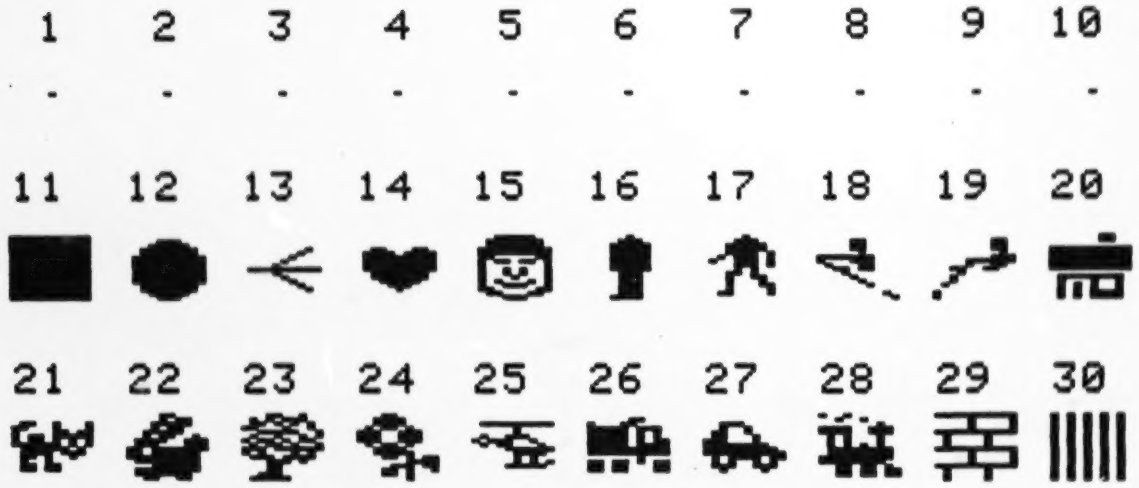
IBM PC jr


②►
Fn B
STOP
STOP
Ctrl D
DOWN
Ctrl F
FLIP
FLIP
Ctrl U
UP
Fn →
PREVIOUS SCREEN
Fn ←
NEXT SCREEN
LOGO Writer
For the IBM PCjr
◄①
Fn 1
SELECT
SELECT
Fn 2
CUT
Fn 3
C
COPY
Fn 4
PASTE
Fn 6
ERASE TO END OF LINE
Fn 8
LABEL
LABEL
Fn 9
TURTLE-MOVE
Fn 0
HELP

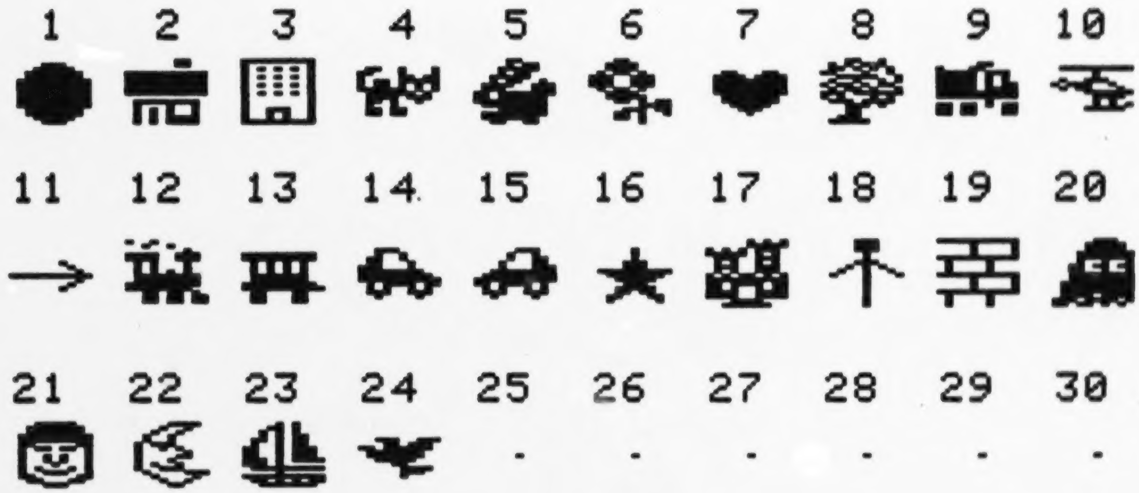
Apple IIe and IIc


②►
S
STOP
STOP
D
DOWN
F
FLIP
FLIP
U
UP
←
PREVIOUS SCREEN
→
NEXT SCREEN
LOGO Writer
For the Apple IIe and IIc
◄①
1
SELECT
SELECT
2
CUT
3
C
COPY
4
PASTE
6
ERASE TO END OF LINE
8
LABEL
LABEL
9
TURTLE-MOVE
0
HELP

APPENDIX 9: Shapes Pages


1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30

INTERMEDIATE


1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30

PRIMARY

APPENDIX 10: Quick Reference

Screen
CC - Clear Command center
CG - Clear Graphics
CP - Clear Page
CT - Clear Text

Scrapbook
CLEARTOOLS
CONTENTS
DOS (IBM only)
ERPAGE *pagename* - ERase Page
FLIP
* FRONT?
GETPAGE (GP) *pagename*
GETSHAPES
GETTOOLS *pagename*
LAST PAGE
LEAVEPAGE
LOAD *pagename*
LOCK
NAMEPAGE (NP) *pagename*
NEWPAGE
* PAGELIST
RESTORE
SAVEPAGE
SHAPES
* TOOLLIST
UNDO
UNLOCK

Workspace
RECYCLE
* SPACE

Pausing
WAIT *number*

Sound
TONE *frequency time*

Input
* BUTTON? *button number*
* KEY?
PADDLE *number*
* READCHAR
* READLIST (RL)
* READLISTCC (RLCC)

Disk
* DISK
SETDISK *letter*

Assigning
CLEARNAME *word/list*
CLEARNAMES
MAKE *name word/list*
NAME *word/list name*
* NAME? *word*
PRINTNAMES
SHOWNAMES
THING *names*

Printer Commands
DSPACE – Double SPACE
PRINTSCREEN
PRINTTEST
PRINTTEXT80
SSPACE – Sincle SPACE

Special Words
END
FALSE
STARTUP
TO
TRUE

Math (Note: all are reporters)
+ - * /
> < =
ARCTAN *number*
COS *degrees*
INT *number*
RANDOM *number*
ROUND *number*
SIN *degrees*
SQRT *number*

Flow of Control/Logic
* AND *true/false1 true/false2*
* IF *true/false list to run*
# IFELSE *true/false list.to.run1 list.to.run2*
* NOT *true/false*
* OR *true/false1 true/false2*
OUTPUT (OP) *word/list*
REPEAT *number list.to.run*
RUN *list.to.run*
STOP
STOPALL
Events
CLEAREVENTS
WHEN *letter list to run*

Graphics

* ALL
# ASK *turtle/turtle.list list.to.run*
BACK (BK) *number*
* BG
CHANGECOLOR *number list of three numbers*
(GS version)
* CHARUNDER - CHARacter UNDER
CLEAN
* COLOR
* COLORUNDER
* COLORVALUE *number* (GS version)
* CURSORPOS
* DISTANCE *list of two numbers* (GS/IBM)
EACH *list.to.run*
FILL
FORWARD (FD) *number*
HEADING
HOME
HT - Hide Turtle
LABEL *word/list*
LEFT (LT) *number*
PD - Pen Down
PE - Pen Erase
* POS
PU - Pen Up
PX - Pen Reverse
RESETCOLORS (GS version)
RG - Reset Graphics
RIGHT (RT) *number*
SETBG *number*
SETC *number*
SETH *number*
SETPOS *[x y]*
SETSH *number*
SETX *number*
SETY *number*
SHADE
* SHAPE
SLOWTURTLE (GS version)
ST - Show Turtle
STAMP
TELL *turtle/turtle.list*
* TOWARDS *list**
* WHO
XCOR
YCOR

Text Editing/Words and Lists

* ASCII *char*
BOTTOM
* BUTFIRST (BF) *word/list*
* BUTLAST (BL) *word/list*
CB - Cursor Back
CD - Cursor Down
CF - Cursor Forward
* CHAR
* CLIPBOARD
COPY
* COUNT
CU - Cursor Up
CUT
DELETE
* EMPTY? *word/list*
EOL
* EQUAL? *word/list1 word/list2*
* FIRST *word/list*
* FOUND?
* FPUT *word/list list*
* IDENTICAL? *word/list1 word/list2*
INSERT *word/list*
* ITEM *number word/list*
* LAST *word/list*
* LIST *word/list1 word/list2*
* LIST? *word/list*
* LPUT *word/list list*
* MEMBER? *word/list1 word/list2*
NEXTSCREEN
* NUMBER?
PARSE *word*
PASTE
PRESCREEN
PRINT (PR) *word/list*
REPLACE *word1 word2*
SEARCH *word*
SELECT
* SELECTED
* SENTENCE (SE) *word/list1 word/list2*
SETEXTPOS *number*
SETTC *number* (GS version)
SHOW *word/list*
SOL - Start Of Line
TAB
* TC (GS version)
* TEXTLEN
* TEXTPOS
TOP
TYPE *word/list*
UNSELECT
* WORD *word1 word2*
* WORD? *word/list*

ProDos Primitives

BYE
COPYFILE *name/pathname1 name/pathname2*
CREATEDIR *pathname*
* DIRECTORIES
ERASEFILE *name/pathname*
* FILELIST
LOADPIC *name/pathname*
LOADTEXT *name/pathname*
* ONLINE
* PREFIX
RENAME *name/pathname1 name/pathname2*
SAVEPIC *name/pathname*
SAVETEXT
SETPREFIX *pathname*
SETSLOT
* SLOT
* .VERSION

System Modifying Primitives
.BLOAD *name/pathname address*
.BSAVE *name/pathname address length*
.CALL *address*
.DEPOSIT *address byte*
* .EXAMINE *address*

IBM DOS Primitives

CHDIR *name/pathname*
COPYFILE *name/pathname1 name/pathname2*
* CURRENTDIR
* DIRECTORIES
DOS
ERASEFILE *name/pathname*
* FILELIST
LOADPIC *name/pathname*
LOADTEXT *name/pathname*
MKDIR *pathname*
RENAME*pathname name/pathname1 name/pathname2*
RMDIR
SAVEPIC *name/pathname*
SAVETEXT
* .VERSION

* Primitive is a reporter (Operation).
# Primitive is either a command or a reporter.

INDEX

" 3, 54
() 9, 54
* 101
+ 101
- 101
/ 101
: 54
< 123
= 123
> 123
[] 7, 9, 54
\ 37, 111

Addition 101
ALL 51
AND 123
Animation 63
  with sound 65
Arithmetic 101
Arrow keys 3
ASCII 70
ASK 86
ASK, EACH TELL
  differences 87
Assignment statement 115

BACK 5
Background, color 11
Backslash 111
Backspace key 23
BASIC 53, 57, 115
Beginning of Line 23
BF 127
BG 67
BK 5
BL 127
Body of procedure 28
BOTTOM 107
Bottom of page 23
Bottom up programming 35
Branches, more than two 83
Branching 77
Bugs 33
BUTFIRST 127
BUTLAST 127
BYE 145

Cartesian commands 119
Cartesian Coordinates 32
Case statements 83
CB 107
CD 107
Centering 24
CF 107
CG 4
CHANGECOLOR 12
CHAR 32, 37, 70
CHDIR 145
CLEAREVENTS 144
Clearing a page, 76
CLEARNAME 116
CLEARNAMES 116
CLEARTOOLS 143
CLIPBOARD 111
CLIPBOARD and SELECTED
  difference 112
COLOR 104
Color 11
Colored text 23
COLORUNDER 104
COLORVALUE 12
Command 54
Command Center
  saving 5
Comments 32
Computer Science Logo Style 1, 55
Conditions, compound 125
Configuring disks 139
CONTENTS 75
Contents Page 3
Control key 143
Coordinates 32
COPY 109
Copy 41ff
Copy keys 41
COPYFILE 145
Copying
  flip side to the front 43
  in the Command Center 42
  procedures 42
  shapes 43
  to another disk 43
  word processor to flip side 43
COS 101
COUNT 131
Counting 93ff
CP 76
CREATEDIR 145
CT 22
CU 107

CURRENTDIR 145
Cursor 3
    Label 8
    losing 9
    moving from procedures 107
CURSORPOS 113
CUT 109
Cut 41ff
Cut keys 41

Debugging 33
    hints and techniques 66, 68, 70, 114, 120,125
Definition, recursive 54
DELETE 109
Delete key 23
Delete/Backspace key 23
Demons 143
Designing programs 35
DIRECTORIES 145
Disk
    Master 3
    Scrapbook 3
DISTANCE 120
Division 101
Documenting procedures 32
DOS 145
Dots 54, 115
DOUBLESPACE 22, 39
Down keys 22
DSPACE 22, 39
Duration 47
Dynamic scoping 115

EACH 85
EACH, TELL, ASK
    differences 87
Editing from procedures 109
Editor 28
Embedded recursion 62, 78ff, 97ff
END 28
End of Line 23
End of line character 44, 108
EOL 107
Equal sign 123
ERASEFILE 145
Erasing
    Label mode 9
    lines 5
    pages 75
    shapes 16
ERPAGE 75

Esc 5, 75
    to leave Label mode 9
    to leave Turtle move 9
    to save shapes 17
Evaluate 53
Events 143
Exchanging files 147
Exporting files 147
Exporting text 24
Extensible 28
False 77, 123, 125

FD 4
FILELIST 146
FILETYPE 146
FILL 19
    problems with 20
FIRST 127
Flashing
    using Label 30
FLIP 76
Flip keys 17, 27
FORWARD 4
FOUND? 112
FPUT 137
Front of page 27
GETPAGE 75
    using 141
GETSHAPES 76
Getting lost 28
Getting started 139
GETTOOLS 143
Global 116
GP 75
Grammar, Logo 53
Graphics, with text 25
GREATER? 125

Harvey, Brian 1, 55, 99
HEADING 120
HOME 52
HT 8

IBM DOS 145
IDENTICAL? 123
IF 67, 77
problems with 68
IFELSE 77
    tips for using 78
Importing files 147
Importing text 24
Incrementing 93ff
    using 96

Indefinite loops 61
Infix notation 102
Inputs
    incrementing 96
    procedure 81
    to pass values 89
    value of 81
INSERT 69
Inserting 23
Interactive 57
Interactive reporters,
    differences 73
    using values 83
    when to use 73
Interface cards 140
ITEM 131

Key
    backspace 23
    delete 23
    down 22
    flip 17, 27
    label 8
    programmable 143
    up 21

LABEL 8
Label Cursor 8
Label keys 8
Label Mode
    leaving 9
    uses for 9
LAST 127
LEFT 5
Less than 123
LESS? 125
Lines
    colored 11
    broken 11
LIST 135
LIST and SENTENCE
    differences 135
List, Logo 53
LIST? 124
LOAD 73, 75
LOADPIC 76
LOADTEXT 76, 146
Local 115
Logo
    grammar 53
    history 1
    list 53
    number 53
    objects 53
    philosophy 1
    punctuation 54
    word 53
LogoWriter Master disk 2
LogoWriter tools 143
Looping 61
Loops,
    definite 67
    indefinite 61
    using inputs 82
Losing work 25
LPUT 137
LT 5

Main procedure 32
MAKE 115
    hints for using 117
    rules for using 116
Margins 24
Master disk 3, 137
Math 101
MEMBER? 124
Middle out programming 35
Mindstorms 1
MKDIR 146
Monochrome screen 11, 12
Moving shapes
    to another disk 44
Moving text,
    from one page to another 43
Mulpiple turtles 51, 85
Multiplication 101
Music toolkit 105
Music, with animation 65

NAMEPAGE 3, 75
New Page 3
NEWPAGE 76
Next Screen 23
NOT 123
Notation,
    infix 102
    prefix 102
Notes
    duration 49
    length 49
    numbers to use 47
NP 3, 75
Number
    Logo 53
NUMBER? 124

Objects
    Logo 53
ONLINE 146
OR 123
OUTPUT 103
    using 105

Page
    clearing 76
    flip side 28
    new 3
    renaming 8
    Shapes 17
PAGELIST 75
Pages
    connecting 141
    erasing 75
Papert , Seymour 1
Parameter passing 90
Parentheses
    in math 102
    with PRINT 38
PARSE 129
Pascal 53, 57, 115
Passing values 83, 89
PASTE 109
Paste 41ff
    keys 41
    shapes 43
Pathnames 145
PD 5
PE 5
PICK 132
Pig Latin 124
Pitch 49
POS 32, 119
POS 119
Predicates 123
PREFIX 146
Prefix notation 102
Previous Screen 23
Primitive 54
PRINT 37, 69
    more than one input 38
Printer, configuring 140
Printing text, problems 22
PRINTNAMES 116
PRINTSCREEN 5, 22
PRINTTEXT 22, 39
PRINTTEXT80 22, 39
Procedure inputs 81
    naming 91
Procedure tree 32
    drawing 33
    example 48
    rules for drawing 33
Procedures 27
    body of 28
    categories 54
    describing 55
    main 32
    moving 43
    multiple 31
    parts of 28
    testing 33
    testing in the Command Center 43
    top level 32
    writing 28
ProDOS 145
Program design 35
Programming style 36
    rules for 36
Programming
    bottom up 35
    finishing 335
    interactive 57
    interactive 71
    middle out 35
    publishing 34
    top down 35
PU 5
Punctuation,
    Logo 54

Quotation mark 3, 54, 67
Quoting 53, 54

RANDOM 11
READCHAR 57
    and READLIST, differences 73
    and READLISTCC, differences 73
    hints for using 58
    saving value 83
READLIST 72
    and READCHAR, differences 73
    saving value 83
READLISTCC 71
    and READCHAR, differences 73
READWORD 72
    saving value 83
READWORDCC 73
Recursion 61
    embedded 62
    embedded 78ff
    embedded 97ff
    tail 61

Recursive call 61
REMAINDER 101
RENAME 146
Renaming a page 8
REPEAT 7
Repetition, indefinite 61
REPLACE 111
Reporter 11, 54
Reporters
  using values 83
  writing 105
RESETCOLORS 12
RESTORE 25, 75
Restoring work
Return character 44, 108
Return/Enter 3
RG 11
RIGHT 4
RMDIR 146
RT 4

SAVEPIC 76, 146
SAVETEXT 146
Saving, Command Center 5
Saving, page 5
Scale, chart 48
Scrapbook disk 3
  making 137
SE 135
SEARCH 111
Search and replace 111
SELECT 109
Select 41ff
Select keys 41
SELECTED 111
  and CLIPBOARD, difference 112
Selecting, on the Shapes page 43
SENTENCE 135
  and LIST, differences 135
SETBG 11
SETC 11
SETH 119
SETPOS 32
SETPOS 119
SETPREFIX 146
SETSH 15
SETSLOT 146
SETTC 23
SETTEXTPOS 113
SHADE 19
  problems with 20
Shape, number of turtle 15
Shapes 15
  copying 43
  copying from one page to another 44
  cutting 43
  defining new 17
  erasing 16
  grid 17
  moving to another disk 41
  page 17
  pasting 43
  restoring 17
  selecting 43
  stamping 15
SHOW 69
SHOWNAMES 116
SIN 101
SINGLESPACE 22, 39
SLOT 146
SOL 107
Sound effects 48
Sound, with animation 65
SPACES procedure 38
Spaces 5
Spaces, inserting 37
SSPACE 22, 39
ST 8
Staff of notes 48
STAMP 15
  problems with 16
Stamping 15
Startup 141
  page 142
  procedure 142
  using 142
Stop
  keys 62
  rule 67
Subprocedures 32
Subtraction 101
Superprocedure 33

TAB 109
Tab key 21
Tail recursion 61
TC 23
TELL 51
  problems with 52
TELL, EACH, ASK
  differences 87
Testing procedures 33
Text manipulation, problems 114

Text,
    centering
    colored 23
    copying 43
    cutting 43
    inserting 23
    pasting 43
    selecting 43
    using 21
    with graphics 25
TEXTLEN 113
TEXTPOS 113
TO 28
TONE 47
    durations 49
    values of notes 47
Tool kit, 103
    music 105
TOOLLIST 143
Tools,
    LogoWriter 143
TOP 107
Top down programming 35
Top level procedure 32
Top of page 23
Tree
    procedure 32
True 77, 123, 125
Turtle Move
    leaving 9
Turtle shape 15
Turtle
    disappearing 16
Turtles, multiple 51
TYPE 69

Underlining 24
UNDO 24
UNSELECT 111
Up keys 21
User defined 54

Values
    keeping 89
    passing 89
    saving 89
Variables 115
    global 116
    local 115
.VERSION 139, 146

WAIT 12, 63
WHEN 143
WHO 52
WORD 135
Word processor 22
Word wrap 22
Word, Logo 53
WORD? 124
Wrapping 7
WRITELN 53

X-coordinate 119
XCOR 120
Y-coordinate 119
YCOR 120

How to increase your Logo Power

Whether you're a Logo teacher, trainer, or enthusiast, you know that this powerful computer language has the potential to have a significant impact on how teachers teach and how students learn. ISTE's Special Interest Group for Logo Educators (SIGLogo) offers you a forum for the exchange of ideas, concepts, and techniques.

What is SIGLogo? SIGLogo is a professional organization that helps Logo Educators get ahead. We sponsor workshops, providing a support community for Logo-using educators. Novice or expert, you will find helpful information in each issue of our journal, *Logo Exchange*.

Satisfaction Guaranteed. Whether you teach Logo or use Logo to teach, SIGLogo and *Logo Exchange* bring you a wealth of ideas from top Logo educators throughout the world, providing you with current information on Logo research, resources, and methods. We're your personal window on professional Logo activities.

Join SIGLogo Today! As a member of SIGLogo, you will receive the *Logo Exchange* journal nine times per year. SIGLogo members are invited to participate in local, regional, and national meetings and to contribute to the flow of ideas through *Logo Exchange*. *Logo Exchange* is published monthly except for June, July, and August. SIGLogo membership is $25 for ISTE members, $30 for non-members. Add an additional $5 for non-U.S. SIGLogo membership.

The International Society for Technology in Education (ISTE) is the leading U.S. and international professional organization for computer educators. It is non-profit, supported by more than 60 organizations of computer using educators worldwide.

ISTE

ISTE/SIGLogo, University of Oregon
1787 Agate Street, Eugene, OR 97403-9905
503/346-4414

INTERNATIONAL SOCIETY FOR TECHNOLOGY IN EDUCATION

ISTE

The *International Society for Technology in Education* touches all corners of the world. As the largest international non-profit professional organization serving computer using educators, we are dedicated to the improvement of education through the use and integration of technology.

Drawing from the resources of committed professionals worldwide, ISTE provides information that is always up-to-date, compelling, and relevant to your educational responsibilities.

Periodicals, books and courseware, *Special Interest Groups, Independent Study* courses, professional committees, and the Private Sector Council all strive to help enhance the quality of information you receive.

Rely on ISTE support:

- *The Computing Teacher* draws on active and creative K-12 educators to provide feature articles and carefully selected columns.
- The *Update* newsletter reaches members with information on the activities of ISTE and its affiliates.
- The *Journal of Research on Computing in Education* comes out with articles on original research project descriptions and evaluations, the state of the art, and theoretical essays that define and extend the field of educational computing.
- Books and courseware enhance teaching materials for K-12 and higher education.
- Professional Committees develop and monitor policy statements on software use, ethics, preview centers, and legislative action.
- The Private Sector Council promotes cooperation between educational technology professionals, manufacturers, publishers, and other private sector organizations.

It's a big world, but with the joint efforts of educators like yourself, ISTE brings it closer. Be a part of the international sharing of educational ideas and technology. Join ISTE.

Join today, and discover how ISTE puts you in touch with the world.

ISTE, University of Oregon
1787 Agate St., Eugene, OR 97403-9905
ph. 503/346-4414

Basic one year membership includes eight issues each of the *Update* newsletter and *The Computing Teacher*, full voting privileges, and a 10% discount off ISTE books and courseware. $28.50

Professional one year membership includes eight issues each of the*Update* newsletter and *The Computing Teacher*, four issues of the *Journal of Research on Computing in Education*, full voting privileges, and a 10% discount off ISTE books and courseware. $55.00